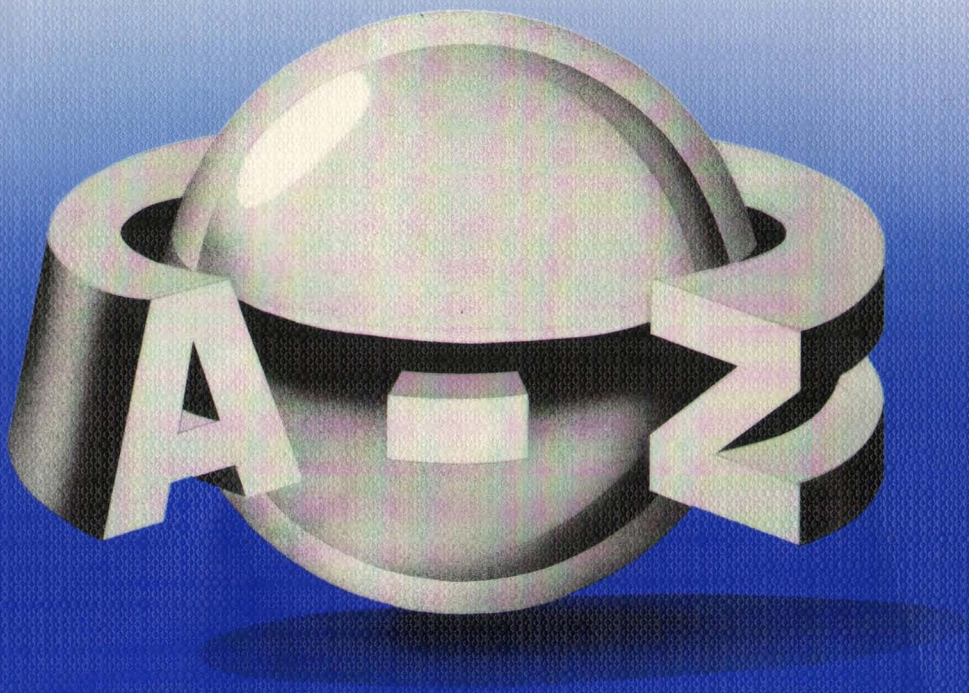
H. Lemcke · V. Dittmar · M. Sommer

Programmier-lexikon für den Atari ST



Hüthig

Hajo Lemcke · Volker Dittmar · Michael Sommer
Programmierlexikon für den Atari ST

Hajo Lemcke · Volker Dittmar · Michael Sommer

Programmierlexikon für den Atari ST

Dr. Alfred Hüthig Verlag Heidelberg

Diejenigen Bezeichnungen von im Buch genannten Erzeugnissen, die zugleich eingetragene Warenzeichen sind, wurden nicht besonders kenntlich gemacht. Es kann also aus dem Fehlen der Markierung ® nicht geschlossen werden, daß die Bezeichnung ein freier Warenname ist. Ebensowenig ist zu entnehmen, ob Patente oder Gebrauchsmusterschutz vorliegen.

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Lemcke, Hajo:
Programmierlexikon für den Atari ST / Hajo Lemcke ; Volker Dittmar ; Michael Sommer. — Heidelberg : Hüthig, 1987.
ISBN 3-7785-1412-1

NE: Dittmar, Volker:; Sommer, Michael:; HST

© 1987 Dr. Alfred Hüthig Verlag GmbH, Heidelberg
Printed in Germany

Vorwort

Neben uns auf dem Schreibtisch, an dem dieses Buch (zum Teil) entstanden ist, liegt ein riesiger Berg Papier. Die englischsprachige (alte) Dokumentation zum Entwicklungspaket ist begraben unter dem Ausdruck von README-Files, die wiederum von aufgeschlagenen Büchern der verschiedensten Art bedeckt sind. Auch andere Dokumentationen, die auf verschiedenen Disketten gespeichert waren, sind dort vorhanden. Der Rest des Tisches, der nicht vom Computer in Beschlag genommen wird, ist mit selbsterstellten Tabellen übersät. Jede Überprüfung wichtiger Informationen führte zum Durchsuchen der Papiere. Stand die Information zum XBIOS nun in Buch X oder in Buch Y? Oder war es Dokumentation Z?

Nun können wir unseren Schreibtisch mal richtig aufräumen. Der Papierberg hat ausgedient, er verstopft jetzt höchstens noch ein entlegenes Schrankfach. Der Grund für unsere Aufgeräumtheit liegt direkt vor Ihnen. Es ist dieses Buch, das die Papierflut ersetzt.

Nachdem bislang alle Informationen nur verstreut zu erhalten waren (oder garnicht), mußten wir das Buch, das wir für unsere alltägliche Programmiertätigkeit brauchen, selbst schreiben. Da auch dieses Buch sicherlich noch Fehler und Unvollständigkeiten enthält, hoffen wir auf rege Kritik von Ihnen.

Da wirkliches Wissen über diesen Rechner sich immer nur auf Teilbereiche beschränkt, haben wir dieses Buch zu dritt geschrieben. Den größten Teil davon hat Volker Dittmar erstellt. Er hat alles beschrieben, was mit VDI und AES zu tun hat. Aber er hat seine Aufgabe mit Bravour gelöst, was Sie als "GEM-Programmierer" sicherlich schnell feststellen werden. Zusätzlich möchten wir unseren Dank **Thies Hammerich** aussprechen, der die Zeichnungen der Schnittstellenbelegung (Anhang C) mit Sachkenntnis und viel Liebe zum Detail gezeichnet hat. Des weiteren bedanken wir uns bei allen, die uns Tips und Hinweise gegeben haben.

Hamburg, November 1986

Die Autoren

Einleitung

Diese Einleitung enthält alle Informationen, die Sie zur optimalen Nutzung dieses Buches benötigen. Das Buch ist aufgebaut wie ein Lexikon. In der Titelzeile einer jeden Seite finden Sie am äußeren Rand entweder das auf dieser Seite in der ersten Zeile beginnende Stichwort oder das von der vorangehenden Seite noch weiterführende. Die Stichworte selbst sind fett, unterstrichen und durch Breitschrift hervorgehoben. Hinweise auf weitere Stichworte werden dadurch kenntlich gemacht, daß dem Wort ein '»' voransteht. Kursiv geschriebene Worte sind entweder Namen von Betriebssystemroutinen oder von »Systemvariablen (Hier haben wir bereits den ersten Verweis auf ein Stichwort).

Wir haben uns auf folgende einheitliche Nomenklatur für die Typendeklaration von Variablen geeinigt. Mit **Byte** wird ein 8 Bit großer Wert bezeichnet, mit **Word** ein 16 und mit **Long** ein 32 Bit großer Wert. Ein **String** ist eine beliebig lange Folge aus ASCII-Zeichen, die mit einem Nullbyte beendet wird. Ein Zeiger auf einen Wert wird durch einen Stern '*' kenntlich gemacht. Jeder Zeiger hat eine Länge von 32 Bit, wobei die Bits 31 bis 24 gelöscht sind, da die CPU im Atari ST nur über einen 24 Bit breiten Adreßbus verfügt. Jeder Zeiger zeigt auf einen Wert eines bestimmten Typs, z.B. ist mit "**char** *" ein Zeiger auf einen String gemeint. Zeiger auf eine Variable in einem Funktionsaufruf werden mit "&" gekennzeichnet (wie in der Sprache C üblich). Über diesen Zeiger kann die aufgerufene Routine Parameter zurückgeben.

Mit **sedezimal** sind Zahlen der Basis 16 gemeint. Diese Zahlen werden durch ein voranstehendes Dollarzeichen '$' von Dezimalzahlen unterschieden und benutzen außer den Ziffern von 0 bis 9 noch die Buchstaben von 'A' bis 'F' für die entsprechenden Dezimalzahlen von 10 bis 15. Diese Zahlen werden auch oft als Hexadezimalzahlen bezeichnet, was aber leider völlig falsch ist, denn Hexa stammt aus dem Griechischen und Dezimal aus dem Latein... Mit **dual** sind Zahlen der Basis Zwei gemeint, d.h. es gibt nur die Ziffern 0 und 1. Diese Zahlen sind durch ein voranstehendes Prozentzeichen '%' kenntlich gemacht. Sie entsprechen der internen Zahlendarstellung des Computers.

Sie werden sich wundern, im Zeitalter der Computer ein Buch zu finden, das ohne Blocksatz erstellt wurde. Aber dabei handelt es sich um volle Absicht. Texte mit Flatterrand lassen sich leichter und schneller lesen (wie verschiedene Untersuchungen gezeigt haben). Sie müssen sich bei Absätzen ohne Randausgleich nicht an ständig wechselnde Wortzwischenräume gewöhnen. Wir halten Flatterrand für lesefreundlicher.

Accessory

Ein A. ist eine Programm, dessen Name auf ".ACC" endet (siehe »Extension) und das beim Booten des System von der Diskette automatisch mitgeladen wird. Das A. bleibt von da an bis zum nächsten Reset stets im Speicher resident. Es läuft permanent im Hintergrund mit (siehe »Multitasking) und wartet im Allgemeinen auf seinen Aufruf. Der Aufruf kann von allen »Anwendungen aus geschehen, die »Drop-Down-Menüs unterstützen. Unter dem linken Menüpunkt in der Menüzeile findet man die Accessories wieder. Klickt man sie an, so wird die Kontrolle an das angeklickte Desk-Accessory übergeben. Dafür ist »AES verantwortlich.

Es gibt bereits eine Fülle von Desk-Accessories, die viele unterschiedliche Funktionen ausführen können. Da gibt es Uhren, Disk-Utilities, Kalender, Datenbanken, Monitore, »Shells, Spiele, Taschenrechner, Notizblöcke, Wecker, Druckerspooler usw. usf., darunter auch einige integrierte Programmpakete, die eine vielzahl von Funktionen vereinigen.

Die Programmierung von A. läßt sich allgemeingültig kaum beschreiben. Die Unterschiede der einzelnen Programmiersprachen sind in diesem Punkt immens (einige Sprachen bzw. Sprachimplementationen unterstützen A. überhaupt nicht). Lesen Sie die Hinweise des Herstellers der Sprache.

ACIA 6850

Abk. für Asynchronous Communications Interface Adapter. Von diesem I/O-Chip sind im Atari ST zwei vorhanden. Deshalb folgt zunächst eine allgemeine Beschreibung dieses Chips und dann deren Verwendung im Atari ST.

Dieser Chip enthält vier Register, von denen zwei nur beschrieben und die beiden anderen nur gelesen werden können.

- Das **Steuerregister** hat eine Länge von 8 Bit, die folgende Funktion haben. Es kann nur beschrieben werden.

Bit 1+0 = Diese beiden Bits bestimmen, durch welchen Faktor die Systemfrequenz für Sende- und Empfangskanal geteilt werden soll:
%00 = Es erfolgt keine Teilung.
%01 = Teilung durch 16 (für die »MIDI Schnittstelle)
%10 = Teilung durch 64 (für den »Tastaturprozessor)
%11 = Chip-Reset. Da dieser Chip keinen Reset-Pin besitzt, muß er vor Benutzung erst softwaremäßig in den Startzustand versetzt werden.

Bit 4-2 = Diese Bits legen das Format der asynchronen Datenübertragung fest:

Wert	Datenbits	Stopbits	Parität	
%000 -	7	2	gerade	
%001 -	7	2	ungerade	
%010 -	7	1	gerade	
%011 -	7	1	ungerade	
%100 -	8	2	keine	
%101 -	8	1	keine	(Beide ACIA's)
%110 -	8	1	gerade	
%111 -	8	1	ungerade	

Bit 6+5 = Festlegung des Pegels am RTS-Pin und der Erlaubnis, einen Interrupt auszulösen, wenn ein Zeichen vollständig gesendet wurde.
%00 = RTS low, kein Interrupt
%01 = RTS low, Interrupt erlaubt
%10 = RTS high, kein Interrupt
%11 = RTS low, kein Interrupt, Break senden
Break senden bedeutet, eine lange Folge (>10) an Einsen zu senden.

Bit 7 = Ist dieses Bit gesetzt, wird ein Interrupt ausgelöst, wenn ein Byte vollständig empfangen wurde, wenn der Empfangspuffer übergelaufen ist oder wenn die DCD-Leitung von low auf high geht. Letzteres kann jedoch im Atari ST nicht passieren, weil der zugehörige Pin fest auf Masse gelegt ist.

- Das **Statusregister** ist ebenfalls 8 Bit breit, kann nur gelesen werden und liefert folgende Informationen:

Bit 0 = Ist dieses Bit gesetzt, dann ist das Empfangsregister voll und muß ausgelesen werden, um einem Überlauf zuvorzukommen.

Bit 1 = Dieses Bit ist gesetzt, wenn der Sendepuffer leer ist und wieder beschrieben werden kann.

Bit 2 = Hat denselben Zustand wie der DCD-Pin, ist im Atari ST also immer Null.
Bit 3 = Hat denselben Zustand wie der CTS-Pin. Da dieser im Atari ST ebenfalls fest auf Masse gelegt ist, hat dieses Bit immer den Wert Null.
Bit 4 = Zeigt einen Frame-Error an. Das bedeutet, daß am Ende eines Bytes kein Stopbit erkannt wurde.
Bit 5 = Zeigt einen Überlauf des Empfangspuffers an. Wird durch Lesen des Empfangspuffers wieder gelöscht.
Bit 6 = Ist die Paritätsprüfung eingeschaltet, zeigt dieses Bit im gesetzten Zustand einen Paritätsfehler an.
Bit 7 = Zeigt an, daß dieser Chip einen Interrupt ausgelöst hat.

- Das **Senderegister** kann nur beschrieben werden und ist ebenfalls acht Bit breit.
- Das **Empfangsregister** kann nur gelesen werden und ist auch acht Bit breit.

Kommen wir zur Beschreibung der ACIA's im Atari ST. Bei beiden ACIA's sind die Pins DCD und CTS fest auf Masse gelegt, so daß die Chips immer sendebereit sind (»V.24-Schnittstelle). Der Pin RTS ist mit nichts verbunden, so daß dessen Programmierung über das Steuerregister ziemlich sinnlos ist.

Die Tastatur-ACIA belegt die Adresse $FFFC00 für das Steuer- und Statusregister und die Adresse $FFFC02 für das Sende- und Empfangsregister. Sie arbeitet mit einer Frequenz von 7812.5 Baud (Bit/Sekunde). Dieser krumme Wert ergibt sich aus den 500 kHz Taktfrequenz und dem Teilungsfaktor von 64. Wie bereits erwähnt, werden die Register dadurch unterschieden, daß entweder gelesen oder geschrieben wird. Die MIDI-ACIA belegt die Adresse $FFFC04 für das Steuer- und Statusregister und die Adresse $FFFC06 für Sende- und Empfangsregister. Die Baudrate dieses ACIA beträgt 500 kHz geteilt durch 16, also 31250 Baud.

AES

Abk. für "Application Environment Services"

AES ist ein Bestandteil von »GEM. Es besteht aus folgenden Komponenten:

1. Einer Sammlung von Unterroutinen für die unterschiedlichsten Zwecke.
2. Dem Multitasking-Kernel, für ein begrenztes »Multitasking.
3. Einer »Shell, um Programme aufzurufen und zu starten.
4. Einem Puffer für »Desk-Accessories.
5. Einem Puffer für »Menü's und »Alarmboxen.

1. Die Sammlung von Unterroutinen (»Libraries) besteht aus folgendem:

   a.) »Application Library,
   b.) »Event Library,
   c.) »Graphic Library,
   d.) »Object Library,
   e.) »Resource Library,
   f.) »Shell Library,
   g.) »Scrap Library,
   h.) »Window Library.

2. Der Kernel unterstützt bis zu sechs »Desk-Accessories, ein Hauptprogramm und den »Screen-Manager (der für Mausfunktionen und »Drop-Down-Menüs zuständig ist). Die »Desk-Accessories laufen permanent im Hintergrund mit, und die Aufgabe des Kernels ist es, die CPU-Zeit zwischen den verschiedenen Prozessen so aufzuteilen, daß möglichst keine Verminderung der Geschwindigkeit festzustellen ist. Dies wird dadurch erreicht, daß die einzelnen Programme AES lediglich mitteilen, auf welche »Events (= Ereignisse) sie warten. Es gibt zwei Listen, die »Not-Ready-List und die »Ready-List. Auf ersterer Liste stehen die Programme, die auf ein Ereignis warten; auf letzterer Liste stehen die Programme, die auf nichts warten und daher laufen. Bei jedem AES-Aufruf wird das oben auf der Liste stehende (gerade laufende) Programm unten auf die Liste gesetzt. Dann wird getestet, ob eines der »Events eingetreten ist und wenn ja wird das Programm, das darauf wartet, auf die »Ready-List gesetzt.

3. Die »Shell lädt und startet das vom Benutzer gewünschte Programm. Sie entscheidet, ob das Programm Text-Orientiert (»TOS-Programme mit der »Extension ".TOS" oder ".TTP") oder Grafik-Orientiert ist und ob Parameter zu übergeben sind (bei ".TTP"). Bei »TOS-Programmen wird der Bildschirm gelöscht, bei »GEM-Programmen nur die »Fenster und »Icons des »Desktop. Bei ".TTP"-Pro-

grammen erscheint eine »Dialogbox, in die der Benutzer die zu übergebenden Parameter eintragen kann.

4. »Desk-Accessories sind »residente Programme. Daher muß AES für sie Speicher reservieren und sie initialisieren. Dies geschieht mit dem Booten.

5. Bei dem Puffer handelt es sich um reservierten Speicherplatz. Dieser wird dazu verwendet, den Hintergrund bei »Drop-Down-Menüs und bei »Alarmboxen zwischenzuspeichern und daraus später wieder auf den Bildschirm zu kopieren. Der Puffer kann ein Viertel des Bildschirminhalts aufnehmen. Dies legt die obere Grenze für die Größe von »Alarmboxen und »Drop-Down-Menüs fest.

AES benötigt einige Array's, um darin globale Variablen ablegen zu können. Diese müssen von der jeweiligen »Application definiert werden. Die Bedeutung der Felder ist folgende:

- GEM-AES Parameter Block:

Offs.	Typ	Bedeutung
0	long	Adresse des control-Arrays (nicht zu verwechseln mit dem contrl-Array für »VDI!)
4	long	Adresse des global-Arrays
8	long	Adresse des int_in-Arrays
12	long	Adresse des int_out-Arrays
16	long	Adresse des addr_in-Arrays
20	long	Adresse des addr_out-Arrays

Das control-Array besteht aus Wörtern. Es wird benutzt, um AES Informationen über die Nummer der angesprochenen Routine zu geben sowie über die Anzahl der Parameter. Der Aufbau ist folgender:

Offs.	Typ	Bedeutung der Werte im control-Array
0	word	Nummer der angesprochenen Routine (op code)
2	word	Anzahl der Parameter (Wörter) im int_in-Array
4	word	Anzahl der Parameter (Wörter) im int_out-Array
6	word	Anzahl der Parameter (Langwörter) im addr_in-Array
8	word	Anzahl der Parameter (Langwörter) im addr_in-Array

Vor dem Aufruf einer AES-Funktion müssen control(0), control(1) und control(3) mit den für die Funktion spezifischen Werten belegt werden. Die Werte werden bei der Beschreibung der AES-Routinen mit angegeben. AES gibt in control(2) die Anzahl der zurückgegebenen Wörter in int_out an, die Anzahl der Langwörter in addr_out steht dann nach dem Aufruf in control(4). Auch diese Werte sind bei der jeweiligen Erläuterung der Routinen mit angegeben.

Weiterhin gibt es noch ein Global Array namens **global**. Es besteht ebenfalls aus Wörtern und wird von **allen** AES-Routinen benutzt. Auch dieses **muß** von der »Application eingerichtet und initialisiert werden.

Offs.	Typ	Bedeutung
0	word	Die Versionsnummer von »GEM
2	word	Die Anzahl der Anwendungen, die von AES gleichzeitig unterstützt werden
4	word	die »Task-ID der »Application (wird von »*appl_init()* zurückgegeben)
6	long	Dieser Wert kann von der »Application nach Belieben verwendet werden
10	long	die Adresse, an die das »Resource-File nach Aufruf von »*rsrc_load()* geladen wurde - also ein Zeiger auf die »Objektbäume
14-30	word	reserviert

Ein Programm, das AES-Routinen aufrufen möchte, muß folgendermaßen vorgehen:

1. Die AES-Arrays müssen aufgebaut werden. Dazu gehören der AES-Parameter-Block, das Control-Array und das Global-Array. Von diesen ist die Länge vorgegeben (siehe oben). Ferner müssen noch folgende vier Arrays vorhanden sein:

int_in:
Integer-Eingabe-Array, alle an AES übergebenen Parameter stehen darin. Die Größe ist nicht vorgegeben, aber es muß ausreichend Platz vorhanden sein. Bei jeder AES-Routine finden Sie Angaben über den benötigten Speicherbedarf. 128 Wörter sind als reichlich zu betrachten.

int_out:
Integer-Ausgabe-Array, hier findet man die Rückgabeparameter von AES wieder. Für die Größe gilt dasselbe wie beim **int_in**-Array.

addr_in:
Adressen-Eingabe-Array, ein Array von Langwörtern. Falls an AES Adressen als Parameter übergeben werden müssen, so werden diese hier eingetragen. 16 Langwörter reichen im allgemeinen aus.

addr_out:
Wenn AES Adressen zurückgibt, so werden diese hier eingetragen. Für die Größe gilt dasselbe wie unter **addr_in** gesagt.

Die Adressen der Arrays müssen in den AES-Parameter-Block eingetragen werden, damit AES sie finden kann.

2. **Bevor** eine AES-Routine benutzt werden kann, muß »*appl_init()* aufgerufen werden. Diese Routine initialisiert die spezifischen Daten-Strukturen und gibt die »Task-ID der Application zurück. Diese ID wird in das Global-Array eingetragen und wird von AES benutzt, um die »Application beim Aufruf identifizieren zu können.

3. Um die entsprechende Routine aufrufen zu können, werden die Parameter in die entsprechenden Arrays eingetragen. Bei der Erläuterung der einzelnen Routinen wird dies jedesmal ausführlich angegeben. Die meisten Hochsprachen (insbesondere natürlich C) bieten ein Interface an, das diese Arbeit dem Programmierer abnimmt. Dies ermöglicht es, die Routinen einfach beim Namen aufzurufen. Assembler-Programmierer müssen dies "von Hand" machen, aber das ist prinzipiell nur Fleißarbeit.

4. Die Informationen, die das Programm benutzt, um »Objekte darzustellen (»Dialogboxen, »Drop-Down-Menüs, »Icons, Text etc.) um mit dem Benutzer zu interagieren, werden in »RSC-Files gehalten. Diese Datei muß geladen werden und die Adressen der Objekte müssen ermittelt werden. Wie dies geschieht, ist exemplarisch unter dem Stichwort »Dialogboxen erläutert. Da es drei verschieden Bildschirmauflösungen gibt, werden auch verschiedene »RSC-Files benötigt. Dazu muß das Programm wissen, welche Auflösung gerade eingeschaltet ist. Dies ist möglich durch einen Aufruf von »*graf_handle()*. Diese Routine gibt den »handle für den Bildschirm

zurück. Dann kann die Routine »*v_opnvwk()* benutzt werden, um eine »virtual Workstation zu öffnen. Beim diesem Aufruf bekommt man - unter anderem - auch die Bildschirmauflösung als Information mitgeteilt.

5. Nun kann mittels »*rsrc_load()* das entsprechende Resource-File geladen werden. Damit hat das Programm Zugriff auf die darin abgespeicherten »Objekte, deren Adressen mit »*rsrc_gaddr()* bestimmt werden können.

6. »GEM ist Ereignis-orientiert (siehe »Event), d. h. daß GEM-Programme auf Aktionen des Benutzers warten und dann angemessene Reaktionen hervorbringen. Meistens geschieht dies durch eine »*evnt_multi()*-Schleife. Dies ist an anderer Stelle genauer beschrieben.

Für welche dieser Schritte der Programmierer verantwortlich ist, hängt von der verwendeten Programmiersprache ab und muß der dazu gehörenden Anleitung entnommen werden.

Auf der Ebene der Maschinensprache sieht ein Aufruf von AES-Routinen so aus:

```
CALL_AES:
        MOVE.L  #aes_parmblk,D1 ; Die Adresse des AES-Parame-
                                ; terblocks wird in das Regi-
                                ; ster D1 geladen
        MOVE.W  #$C8,D0         ; Kennung, daß eine AES-Rou-
                                ; tine aufgerufen werden soll

        TRAP    #2              ; GEM-Funktion aufrufen
```

Der AES-Parameterblock muß natürlich mit den korrekten Werten vorbesetzt sein. Welche der AES-Routinen aufgerufen wird, hängt von 'control(0)' ab (s. o.).

Im Anhang A finden Sie vier Beispiele für einen AES-Aufruf in Basic, Assembler und C.

Alarmbox

Die A. wird benutzt, um den Benutzer auf einen Vorgang aufmerksam zu machen (der z. B. nicht mehr rückgängig zu machen ist). Wenn Sie z. B. im »Desktop den Menüpunkt "Formatieren" anklicken, so erscheint in der Bildschirmmitte erst eine A. Diese weist Sie darauf, daß Sie im Begriff sind, alle Daten auf Ihrer Diskette zu löschen. Falls Sie irrtümlich diesen Menüeintrag gewählt haben, so können Sie "ABBRUCH" wählen (indem Sie den entsprechenden »Button anklicken (oder <Return> drücken). Bei manchen Vorgängen geben A. auch nur Hinweise auf Fehler (z. B.: "Datei nicht gefunden"). Diese haben nur einen »Button. Mit anderen Worten: A. sind einfache »Dialogboxen.

Folgende Routinen können zur Darstellung von A. verwendet werden:

-»*form_alert()*:
Dient der Darstellung von »Alarmboxen - dort finden Sie auch noch weitere Erläuterungen zu den Alarmboxen

-»*form_error()*:
Alarmbox für »DOS-Fehler

Wie diese Aufrufe zu geschehen haben, entnehmen Sie bitte den Beschreibungen der Routinen. Vorbedingung für die Verwendung von A. ist, daß es sich um ein »GEM-Programm handelt. »TOS-Programme können keine A. verwenden.

A. ersetzen auf modernen PC's die aus der Computer-Steinzeit stammenden Fehlermeldungen wie etwa: "Fatal Error Nr. $4E -Press Return".

Anwendung

siehe »Application

APPBLK

Eine Struktur, die vom Programmierer definierte »Objekte verfügbar macht. Die Struktur ist unter »Objekte im Zusammenhang erklärt. Sie wird für vom Programmierer definierte »Objekte benötigt.

Appl exit

Zur »Application-Library gehörige Routine:

```
$13 = 19        ap_xreturn = appl_exit()
word            ap_xreturn;
```

Hiermit wird AES dazu veranlaßt, das Anwenderprogramm von seiner Liste zu löschen, die ID-Nummer (»Task-ID) freizugeben und die Datenfelder "aufzuräumen". Nach diesem Routinenaufruf dürfen keine »GEM- oder »AES-Funktionen mehr aufgerufen werden.

Die Routine ist für ein geordnetes Verlassen des Programms erforderlich. Wird beim Beenden des Programms diese Routine nicht aufgerufen, so nimmt »AES an, daß diese »Application noch läuft. Dann kann kein weiteres »GEM-Programm mehr gestartet werden.

Eingabeparameter: keine

```
contrl(0)   =  19
contrl(1)   =   0
contrl(3)   =   0
```

Ausgabeparameter:

```
word ap_xreturn     0 = Fehler, alles andere: OK

contrl(2)   =   1
contrl(4)   =   0

int_out(0)  =  ap_xreturn
```

Appl find

Zur »Application-Library gehörige Routine:

```
$0D = 13     ap_fid = appl_find(ap_fpname)
byte *                          ap_fpname;
```

Herausfinden einer anderen »Task-ID eines parallel im Speicher laufenden Programms (z. B. eines »Desk-Accessories). Diese Nummer wird benötigt, um mittels »*appl_write()* Nachrichten an das dazugehörige Programm zu schicken.

Eingabeparameter:

byte *fpname	Zeiger auf den (Datei-) Namen der gesuchten »Application. Der Name **muß** genau aus acht Buchstaben bestehen.

```
contrl(0)   =  13
contrl(1)   =   0
contrl(3)   =   1

addr_in(0)  =  ap_fpname
```

Ausgabeparameter:

word ap_fid	Die »Task-ID-Nummer der gesuchten Anwendung. Falls »AES das Programm nicht finden konnte, wird -1 zurückgegeben.

```
contrl(2)   =   1
contrl(4)   =   0

int_out(0)  =  ap_fid
```

Appl_init

Zur »Application-Library gehörige Routine:

```
$0A = 10     ap_id = appl_init()
word         ap_id;
```

Diese Routine wird verwendet, um eine Anwendung (=»Application) anzumelden und zu initialisieren. Sie sorgt dafür, daß »AES die benötigten Datenfelder zur Verfügung stellt und gibt die »Task-ID der Anwendung zurück. Diese Routine muß ganz zu Anfang eines GEM-Programms aufgerufen werden. Die Initialisierungsschritte, die dem Aufruf vorausgehen müssen, sind unter dem Begriff »AES näher erläutert.

Eingabeparameter: keine

```
control(0)  =  10      Opcode
control(1)  =  0       keine Übergabeparameter in int_in
control(3)  =  0       kein Wert in addr_in
```

Ausgabeparameter:

```
word ap_id             Die Identifikationsnummer (Task-ID) der
                       Anwendung

control(2)  =  1       ein Rückgabeparameter in int_out: ap_id
control(4)  =  0       kein Wert in addr_out

int_out(0)  =  ap_id
```

Appl_read

Zur »Application-Library gehörige Routine:

```
$0B = 11    ap_rreturn = appl_read(ap_rid, ap_rlength,
word        ap_rreturn,            ap_rid, ap_rlength;
                                   ap_rpbuff)
word *                             ap_rpbuff;
```

Diese Routine liest eine Anzahl von Bytes aus einer sog. Pipe. Das ist nichts weiter als ein »Messagebuffer, in den eine »Application Mitteilungen hineinschreiben kann. Diese Routine muß dann verwendet werden, wenn die Nachricht länger ist als der von AES dafür vorgesehene Puffer. Der "Sender" schreibt mittels »*appl_write()* die Nachricht.

Diese Funktion bietet eine bequeme Methode, mit der Programme, die parallel im Speicher vorhanden sind, Informationen austauschen können.

Eingabeparameter:

word ap_rid	Die »Task-ID-Nummer der Anwendung, deren Nachricht gelesen werden soll. Die »Task-ID wird von »*appl_init()* vergeben. Gewöhnlich (aber nicht immer!) schreiben »Applications an sich selbst.
word ap_rlength	Anzahl der zu lesenden Bytes.
word *ap_rpbuff	Adresse des Puffers, in den »AES die Nachricht hineinschreiben soll.

```
control(0)  =  11      Opcode
control(1)  =   2
control(3)  =   1

int_in(0)   =  ap_rid
int_in(1)   =  ap_rlength

addr_in(0)  =  ap_rpbuff
```

Ausgabeparameter:

```
word ap_rreturn          0 = Fehler, alles andere: OK

control(2)  =   1
control(4)  =   0

int_out(0)  =  ap_rreturn
```

Appl_tplay

Zur »Application-Library gehörige Routine:

```
$0E = 14    ap_tpreturn = appl_tplay(ap_tpmem, ap_tpnum,
word        ap_tpreturn,                       ap_tpnum;
word *                                ap_tpmem;
                                      ap_tpscale)
word                                  ap_tpscale;
```

Diese Routine ist das Gegenstück zu »*app_trecord()*. Sie spielt

die aufgezeichneten »Events (siehe »*app_trecord()*) wieder ab. Dabei kann sogar die Geschwindigkeit eingestellt werden, mit denen »GEM die Aktionen des Benutzers wiederholt.

Eingabeparameter:

word *ap_tmem	Zeiger auf den Speicherbereich, in dem die abzuspielenden Events stehen
word ap_tpnum	Anzahl der abzuspielenden Events
word ap_tpscale	Der Geschwindigkeitsfaktor (1 bis 10000), mit dem das Abspielen vor sich gehen soll. Dabei bedeuten z. B.: 50 = Halbe Geschwindigkeit 100 = Volle Geschwindigkeit 200 = Doppelte Geschwindigkeit usw.

```
contrl(0)   =  14
contrl(1)   =   2
contrl(3)   =   1

int_in(0)   =   ap_tpnum
int_in(1)   =   ap_tpscale

addr_in(0)  =   ap_tpmem
```

Ausgabeparameter:

word ap_tpreturn	Ist immer gleich eins.

```
contrl(1)   =   1
contrl(4)   =   0

int_out(0)  =   ap_tpreturn
```

Appl_trecord

Zur »Application-Library gehörige Routine:

```
$0F = 15    ap_trreturn = appl_trecord(ap_trmem, ap_trcount)
word        ap_trreturn,                         ap_trcount;
word *                                  ap_trmem;
```

Diese Routine zeichnet alle Interaktionen eines Benutzers mit »GEM auf. Folgende Events werden erfaßt:

- Timer-Event (Verstreichen einer bestimmten Zeit)
- Button-Event (Drücken eines Mausknopfes)
- Mouse-Event (Der Mauspfeil verläßt oder erreicht ein bestimmtes Gebiet)
- Keyboard-Event (Irgendeine Taste wurde gedrückt)

Zur genauen Definition des Begriffs »Event (= Ereignis) siehe auch die Beschreibung der »Event-Routinen.

Zum Abspeichern eines Events werden 6 Bytes benötigt:
- ein Wort um die Art des Events und
- ein Langwort um damit zusammenhängende Werte zu erfassen.

Die Bedeutung des Wortes:
0 = Timer-Event
1 = Button-Event
2 = Mouse-Event
3 = Keyboard-Event

Was das Langwort bedeutet, hängt von dem Event ab:

- Timer-Event:
Anzahl der verstrichenen Millisekunden

- Button-Event:
Die unteren 16 Bit geben an, ob und welcher »Mausknopf gedrückt wurde (Bit gesetzt: zugehöriger Knopf wurde gedrückt), die oberen 16 Bit enthalten die Anzahl der Maus-Klicks.

- Mouse-Event:
Untere 16 Bit enthalten die X-Koordinate der »Maus, die oberen 16 Bit die Y-Koordinate

- Keyboard-Event:
Die unteren 16 Bit enthalten den »ASCII-Code der gedrückten Taste, die oberen 16 Bit enthalten den Code für die gedrückte Shift- Alternate- Control und Capslocktaste (siehe auch die »BIOS- Routine »*Kbshift()*).

Diese Routine ist z. B. für Demo-Programme nützlich, die ohne äußeres Zutun zeigen, was man mit dem Programm anstellen kann (es gibt nämlich noch die Routine »*appl_tplay()*, mit der man diese Aufzeichnungen wieder ablaufen lassen kann).

Eingabeparameter:

word *ap_trmem	Die Adresse der Speicherregion, in der die ganzen »Events abgespeichert werden.
word ap_trcount	Anzahl der aufzuzeichnenden »Events. Da jedes »Event 6 Bytes Speicher braucht, ist das der zur Verfügung stehende Speicher (in Bytes) durch 6 dividiert.

```
contrl(0)   =  15
contrl(1)   =   1
contrl(3)   =   1

int_in(0)   =   ap_trcount

addr_in(0)  =   ap_trmem
```

Ausgabeparameter:

word ap_trreturn	Die Anzahl der von GEM aufgezeichneten »Events

```
contrl(2)   =   1
contrl(4)   =   0

int_out(0)  =  ap_trreturn
```

Appl_write

Zur »Application-Library gehörige Routine:

```
$0C  = 12   ap_wreturn = appl_write(ap_wid, ap_wlength,
word        ap_wreturn,             ap_wid, ap_wlength;
                                    ap_wpbuff)
word *                              ap_wpbuff;
```

Diese Routine ist das Gegenstück zu »*appl_read()*: sie schreibt die angegebene Anzahl von Bytes als Nachricht an eine andere Applikation (oder an sich selbst!). Der Empfänger liest diese Nachricht mit der Routine »*appl_read()*. So ist es z. B. möglich, daß ein Programm mit einem »Desk-Accessory kommuniziert.

Eingabeparameter:

word ap_wid	Die »Task-ID-Nummer der Application, an die die Nachricht geschickt werden soll
word ap_w_length	Anzahl der abzuschickenden Bytes
word *ap_wpbuff	Zeiger auf den Puffer, in dem die Nachricht steht

```
contrl(0)   =  12
contrl(1)   =   2
contrl(3)   =   1

int_in(0)   =  ap_wid
int_in(1)   =  ap_wlength

addr_in(0)  =  ap_wpbuff
```

Ausgabeparameter:

word ap_wreturn 0 = Fehler, alles andere: OK

```
contrl(2)   =   1
contrl(4)   =   0

int_out(0)  =  ap_wreturn
```

Application

Als A. werden Anwenderprogramme bezeichnet, die man in den Speicher laden und ausführen kann. Haben diese die »Extension ".APP" oder ".PRG", dann handelt es sich um Programme, die »VDI und/oder »AES benutzen. Jede dieser A. wird durch eine Nummer (»Task-ID) identifiziert, da sich mehrere A.'s gleichzeitig im Speicher befinden können. Lautet die »Extension auf ".TOS" oder ".TTP", dann hat man Text-orientierte Programme vor sich (»TOS), die weder »VDI noch »AES benutzen. Das heißt nicht, daß TOS-Programme gänzlich auf Grafik verzichten müssen. Wozu gibt es schließlich die »Line-A-Befehle. Allerdings haben »TOS-Programme keine »Task-ID, da nur eines dieser Programme zur Zeit im Speicher laufen kann (jedenfalls mit dem TOS-Betriebssystem).

Application Environment Service

Siehe »AES

Application Library

Bezeichnung für eine Reihe von Unterprogrammen, die zu »AES gehören. Diese Routinen können von jeder »Application benutzt werden, von Programmen und »Desk-Accessories. Die A. umfaßt folgende Routinen:

- *»appl_init()*: Dient der Vorbereitung von weiteren Aufrufen der Application
- *»appl_exit()*: Zum Verlassen der »Application.
- *»appl_find()*: »Task-ID einer anderen »Application finden.
- *»appl_read()*: »Message-Buffer lesen.
- *»appl_tplay()*: Benutzereingaben wieder abspielen.
- *»appl_trecord()*: Benutzereingaben aufnehmen und speichern.
- *»appl_write()*: »Message in »Message-Buffer schreiben.

Zur genaueren Information bitte die entsprechenden Stichworte nachschlagen.

ASCII

Engl.: American Standard Code for Information Interchange. Diese Vereinbarung ordnet einem Byte genau ein Zeichen zu. Diese Zuordnung können Sie der folgenden Tabelle entnehmen.

	$.0	$.1	$.2	$.3	$.4	$.5	$.6	$.7	$.8	$.9	$.A	$.B	$.C	$.D	$.E	$.F
$0.	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
$1.	DLE	XON	DC2	XOF	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
$2.		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
$3.	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
$4.	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
$5.	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
$6.	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
$7.	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Die Codes von $00 bis $1F sind Sonderzeichen, die für asynchrone Datenübertragung von Bedeutung sind. Zum Teil werden sie jedoch auch allgemein verwendet. Sie haben folgende Bedeutung:

$00 = NUL Füllzeichen. Wird teilweise zur Aufrechterhaltung der Leitung verwendet, wenn keine Daten zu senden sind.

$01 = SOH (Start Of Heading) Zeigt den Beginn eines Headers an.

$02 = STX (Start of TeXt) Beginn des Textes.

$03 = ETX (End of TeXt) Ende des Textes.

$04 = EOT (End Of Transmission) Ende der Übertragung.

$05 = ENQ (ENQuiry) Sendeanforderung.

$06 = ACK (ACKnowledge) Bestätigung der Sendeanforderung.

$07 = BEL (BELl) Läßt einen Piepton erklingen.

$08 = BS (BackSpace) Löscht das zuletzt übertragene Zeichen.

$09 = HT (Horizontal Tab) Vorschub zur nächsten Tabulatorposition. Dient zur Ansteuerung von Bildschirmen und Druckern.

$0A = LF (Line Feed) Zeilenvorschub. Setzt den Cursor bzw. den Druckkopf eine Zeile tiefer in dieselbe Spalte.

$0B = VT (Vertical Tab) Vorschub zur Zeile, die den nächsten vertikalen Tabulator enthält.

$0C = FF (Form Feed) Seitenvorschub.

$0D = CR (Carriage Return) Bringt den Cursor bzw. Druckkopf wieder an den Anfang der Zeile. Um an den Anfang der nächsten Zeile zu gelangen, müssen die zwei Zeichen CR ($0D) und LF ($0A) gesendet werden.

$0E = SO (Shift Out)

$0F = SI (Shift In)

$10 = DLE (Data Link Escape) Rückkehr aus der Datenverbindung in die Steuerung des Verbindungsaufbaus.

$11 = XON (Transmission ON) Sagt der anderen Station, daß sie jetzt weiter senden darf (auch DC1 genannt).

$12 = DC2 (Device Control 2) Steuerzeichen, daß je nach angeschlossenem Gerät unterschiedlich ausgewertet wird.
$13 = XOF (Transmission OFf) Die andere Station soll mit dem Senden zunächst aufhören (auch DC3 genannt).
$14 = DC4 (Device Control 4) wie DC2.
$15 = NAK (Negative AcKnowledgement) Wird gesendet, wenn die Anforderung nicht ausgeführt werden konnte.
$16 = SYN (SYNchronization) Dient zum Synchronisieren der Übertragung.
$17 = ETB (End of Transmission Block) Ende eines Blockes.
$18 = CAN (CANcel) Hebt die letzte Anforderung wieder auf.
$19 = EM (End of Medium) Zeigt im allgemeinen das aktuelle Ende der Daten auf einem Band an.
$1A = SUB (SUBstitute)
$1B = ESC (ESCape) Die Zeichen, die diesem Byte folgen, haben bestimmte Steuerfunktionen, z.B. die EPSON-Escape-Sequenzen zur Druckeransteuerung.
$1C = FS (File Separator) Trennt verschiedene »Files auf einem kontinuierlichen Medium wie einem Band.
$1D = GS (Group Separator) Trennt einzelne Gruppen, wie z.B. »Subdirectories, voneinander.
$1E = RS (Record Separator) Trennt einzelne Records innerhalb eines Files.
$1F = US (Unit Separator) Schaltet zwischen verschiedenen Einheiten oder Geräten um.
$7F = DEL (DELete) Löschen des zuletzt übertragenen Zeichens.

Assign.sys

Dies ist der Name einer Datei, die von »GDOS gebraucht wird. Für eine genauere Beschreibung siehe »GDOS.

Base-Page

Wenn ein Programm mit der »GEMDOS-Funktion *PEXEC()* ($4B=75) geladen wird, dann wird vor dem Programm ein 256 Byte großer Speicherbereich angelegt, in dem die B. aufgebaut wird. Diese hat folgenden Aufbau:

$00 = 0	p_lowtpa	Zeiger auf den Anfang der »TPA.
$04 = 4	p_hitpa	Zeiger auf das Ende des freien Speichers. Dorthin wird der User-Stack-Pointer gesetzt, bevor ein mit *PEXEC()* geladenes Programm gestartet wird.
$08 = 8	p_tbase	Zeiger auf das Text-Segment (den Programmcode). Dieser liegt genau 256 Bytes hinter der B.
$0C = 12	p_tlen	Länge des Text-Segments (des Programmcodes) in Bytes.
$10 = 16	p_dbase	Zeiger auf den Beginn des Daten-Segments (dort stehen die initialisierten Daten).
$14 = 20	p_dlen	Länge des Daten-Segmentes in Bytes.
$18 = 24	p_bbase	Zeiger auf den Beginn des BSS-Segments. Dort befinden sich die nicht initialisierten Daten, wie z.B. der Stack dieses Programmes. Das BSS-Segment liegt hinter dem Text- und dem Daten-Segment.
$1C = 28	p_blen	Länge des BSS-Segments in Bytes. Diese Länge wird dem »Programmheader beim Laden entnommen.
$20 = 32	*p_xdta	Disk-Transfer Adresse.
$24 = 36	*p_parent	Zeiger auf die B. des aufrufenden Programmes.
$28 = 40	p_lddrv	Nummer des Laufwerks, von dem dieses Programm geladen wurde.
$29 = 41	p_curdrv	Nummer des aktuellen Laufwerkes für dieses Programm.
$2A = 40		reserviert.
$2C = 44	p_env	Zeiger auf den »Environment-String.
$30-$7F		Dieser Bereich wird vom Betriebssystem verwendet und sollte vom neu geladenen Programm nicht benutzt werden.
$80 = 128	cmd_line	Hier werden die aktuellen Parameter eingetragen, die beim Aufruf des Programmes angegeben wurden.

Benutzeroberfläche

Als B. bezeichnet man denjenigen Programmteil, mit dem der Benutzer des Programms unmittelbaren Kontakt hat. Dazu gehören z. B. physikalische Geräte wie etwa Monitor, »Maus oder Tastatur, aber vor allem die Art und Weise des Ansprechens des Programms (Bedienerführung).

Der Atari ST stellt als Benutzeroberfläche *nur* die mit der Maus steuerbare Grafikoberfläche zur Verfügung. Um einem Programm Parameter übergeben zu können, muß dieses als ".TTP"-Programm deklariert sein. Eine andere Art der Benutzeroberfläche stellen die sogenannten »Kommandointerpreter dar.

Bildschirm

Siehe »Bildschirmspeicher und »Bildschirmsteuerung

Bildschirmspeicher

Der B. des Atari ST benötigt in allen drei Auflösungsstufen einen Speicher von 32000 Byte, dessen jeweiliger Aufbau jedoch von der aktuellen Auflösungsstufe abhängt. In der Stufe A2 (640 * 400 Punkte) entspricht jedem Bit des Speichers genau ein Pixel auf dem Bildschirm. Der Pixel in der linken oberen Ecke des Schirmes wird durch Bit 15 des ersten Wortes des B.s repräsentiert. Das zweite Pixel durch Bit 14, usw. Für jede Zeile werden 640/16, also 40 Worte benötigt. Für die 400 Pixel Auflösung in der Senkrechten werden somit 400*40 gleich 16000 Worte bzw. 32000 Byte benötigt (nicht zu verwechseln mit 32 kByte).

In der Auflösungsstufe A1 (640 * 200 Punkte) stellen die korrespondierenden Bits von je zwei aufeinanderfolgenden Worten einen Pixel auf dem Bildschirm dar. Der Pixel in der linken oberen Ecke des Schirmes wird durch die Bits 15 der beiden ersten Worte dargestellt. Diese zwei Bits legen eine von vier möglichen Farben fest, die in den »Farbregistern angegeben werden. Die Bits 14 bestimmen die Farbe des zweiten Punktes in der obersten Zeile, usw. Für eine Zeile werden jetzt 2*640/16=80 Worte benötigt. Die zweite Zeile beginnt nun erst mit dem 81. Wort des B.s.

In der farbigsten Stufe (A0) mit einer Auflösung von 320 * 200 Punkten legen je vier aufeinanderfolgende Worte die Farben von 16 Pixeln auf dem Bildschirm fest. Mit vier Bit kann eine von 16 möglichen Farben ausgewählt werden. Die Farbe des Pixels in der linken oberen Ecke wird durch die Bits 15 der ersten vier Worte des B.s bestimmt. Da die horizontale Auflösung gegenüber der Stufe A1 halbiert wurde, jetzt aber vier Worte je Pixel benötigt werden, wird eine Zeile wiederum durch 80 Worte dargestellt.

Bildschirmsteuerung

Für grafische Darstellungen werden zur B. im allgemeinen die »Line-A Befehle benutzt. Je nach Anwendung werden diese jedoch nicht direkt aufgerufen, sondern man benutzt die Routinen aus den »VDI- oder »AES-Bibliotheken.

Für textorientierte Darstellungen können mit Hilfe der »GEMDOS-Funktionen 2,6 oder 9 die normalen ASCII-Zeichen, mit der »BIOS-Funktion 3 bei Angabe von Devicetyp Nummer fünf alle ASCII-Zeichen und mit allen Routinen zusätzliche Steuerzeichen ausgegeben werden. Diese Steuerzeichen umfassen zum einen die üblichen ASCII-Zeichen wie Return (13), Backspace (8), usw. und zum anderen die vom Video-Terminal 52 (VT52) her bekannten Escape-Sequenzen. Diese Sequenzen beginnen mit dem Steuerzeichen Escape (27=$1B) dem ein normales ASCII-Zeichen folgt:

ESC A - Cursor up
Cursor eine Zeile nach oben bewegen. In Zeile Null geschieht nichts.

ESC B - Cursor down
Cursor eine Zeile nach unten bewegen. In der untersten Zeile geschieht nichts.

ESC C - Cursor right
Cursor eine Spalte nach rechts bewegen. Je nachdem ob das »Wrapping ein- oder ausgeschaltet ist, erfolgt in der ganz rechten Spalte ein Übergang auf den Beginn der nächsten Zeile, oder es geschieht nichts.

ESC D - Cursor left
Cursor eine Spalte nach links. In Spalte Null geschieht nichts.

ESC E - Clear-Home
Bildschirm löschen und Cursor in linke obere Ecke setzen.

ESC H - Cursor Home
Cursor in linke obere Ecke setzen.

ESC I - Scroll down
Cursor eine Zeile nach oben bewegen. Befindet sich der Cursor in der obersten Zeile, wird dort eine Leerzeile eingefügt und die restlichen Zeilen werden nach unten gescrollt.

ESC J - Clear to end of screen
Der Bildschirm wird ab der aktuellen Cursorposition einschließlich bis zum Ende gelöscht.

ESC K - Clear to end of line
Der Rest der Zeile wird ab der aktuellen Cursorposition einschließlich gelöscht.

ESC L - Insert line
Über der aktuellen Zeile wird eine Leerzeile eingefügt. Die folgenden Zeilen werden nach unten gescrollt. Danach befindet sich der Cursor am Anfang der Leerzeile.

ESC M - Delete line
Die aktuelle Zeile wird gelöscht und der Rest des Bildschirms wird nachgerückt. Der Cursor befindet sich dann in Spalte Null der nachgerückten Zeile.

ESC Y - Set cursor
Mit diesem Befehl kann der Cursor frei auf dem Bildschirm positioniert werden. Dazu müssen zwei weitere Parameter folgen, die die Zeile und die Spalte angeben. Die linke obere Ecke ist Zeile Null, Spalte Null. Beide Werte müssen mit einem Offset von 32 übergeben werden. Um den Cursor z.B. in Zeile 10, Spalte 40 zu setzen, müssen folgende vier Werte ausgegeben werden: $1B, $59, $2A, $48.

ESC b - Set colour for characters
Hiermit kann die Farbe bestimmt werden, in der alle folgenden Zeichen ausgegeben werden. Bei einem Monochrommonitor gibt es nur die Werte 0 für Weiß und 1 für Schwarz. In der Auflösungsstufe A1 stehen 4 Farben (0-3) und in A0 stehen 16 Farben (0-15) zur Verfügung.

ESC c - Set colour for background
Mit diesem Befehl kann entsprechend die Hintergrundfarbe für die nachfolgenden Zeichen bestimmt werden.

ESC d - Clear from start of screen
Löscht den Bildschirm vom Anfang bis zur aktuellen Cursorposition einschließlich.

ESC e - Cursor on
Schaltet den Cursor ein. Mit »XBIOS 21 können weitere Funktionen bestimmt werden.

ESC f - Cursor off
Schaltet den Cursor wieder aus.

ESC j - Save Cursorposition
Speichert die aktuelle Position des Cursors. Diese Funktion wird auch von anderen Escape-Sequenzen benutzt, so daß in einem solchen Fall Vorsicht geboten ist.

ESC k - Restore Cursor
Setzt den Cursor wieder an die mit "ESC j" gespeicherte Position zurück. War nichts gespeichert, wird der Cursor an die Home-Position (linke obere Ecke) gesetzt.

ESC l - Clear line
Löscht die Zeile, in der sich der Cursor befindet. Danach steht der Cursor am Beginn dieser Leerzeile.

ESC o - Clear from start of line
Löscht alle Zeichen vom Beginn der aktuellen Zeile bis zur Cursorposition.

ESC p - Reverse on
Schaltet den Reversmodus ein. Bei allen folgenden Zeichen sind Vorder- und Hintergrundfarbe vertauscht.

ESC q - Reverse off
Schaltet den Reversmodus wieder aus. Vorder- und Hintergrundfarbe nehmen wieder ihre alten Werte an.

ESC v - Wrap on
Schaltet den automatischen Zeilenüberlauf ein. Wird ein normales ASCII-Zeichen ausgegeben, wenn sich der Cursor in der letzten Spalte befindet, wird dieses automatisch am Anfang der nächsten Zeile ausgegeben.

ESC w - Wrap off
Schaltet den automatischen Zeilenüberlauf wieder aus.

Bios

Abk. für -Basic Input Output System- ist die unterste Ebene des Betriebssystems über die alle Peripheriegeräte angesprochen werden können. Das Bios wird über »Trap #13 angesprochen. Alle Fehler und Rückgabewerte enthält das Register D0. Ein Assemblerauf-ruf sieht folgendermaßen aus:

```
move.w #$01,-(sp)    * RS 232
move.w #$01,-(sp)    * Funktionsnummer von Bconstat()
trap   #13           * Bios aufrufen
addq.l #$04,sp       * Stack reparieren
tst.w  D0            * ist ein Zeichen vorhanden
bne    hole_zeichen  * wenn ja, verzweigen
```

Bei Assemblerprogrammierung muß darauf geachtet werden, daß der Trap-Handler nicht die Register D0-D2 und A0-A2 sichert.

Kurzübersicht der Funktionen des BIOS mit Parametern:

Funktionsnummer	Funktion mit Parameter
$00 = 0	Getmpb(p_mpb)
$01 = 1	Bconstat(dev)
$02 = 2	Bconin(dev)
$03 = 3	Bconout(dev,c)
$04 = 4	Rwabs(rwflag,buf,count,recno,dev)
$05 = 5	Setexc(vecnum,vec)
$06 = 6	Tickcal()
$07 = 7	Getbhp(dev)
$08 = 8	Bcostat(dev)
$09 = 9	Mediach(dev)
$0A = 10	Drvmap()
$0B = 11	Kbshift(mode)

Das Bios kann folgende Geräte ansprechen:

Gerätenummer	Name	Beschreibung
0	PTR:	Drucker (»Centronicsschnittstelle)
1	AUX:	»RS 232 Schnittstelle
2	CON:	»Bildschirm, »Tastatur
3	MIDI:	»Midischnittstelle
4	KBD:	»Tastaturprozessor
5	SCR:	»Bildschirm

Erlaubte Geräte bei folgenden Aufrufen:

Aufruf	PRT	AUX	CON	MIDI	KBD	SCR
Bconstat()	nein	ja	ja	ja	nein	nein
Bconin()	ja	ja	ja	ja	nein	nein
Bconout()	ja	ja	ja	ja	ja	ja
Bcostat()	ja	ja	ja	ja	ja	nein

Warnung:
Sie sollten beachten nicht Langwörter zu testen, wenn ein Bios-aufruf einen Fehler liefert, da die oberen 16 Bits mit irgend-welchen Werten gefüllt sein können, d.h. testen Sie nur Wörter.

Hinweis:
Das Bios läßt sich im Gegensatz zum Gemdos bei Ein/Ausgabe nicht mit Control C, Control S und Control Q beeinflussen.

$00 = 0 **Getmpb(p_mpb)**
long **p_mph;**

Es p_mpb muß ein Zeiger sein, der auf eine Struktur zeigt der für drei Langwörter reserviert ist. Dieser Block wird mit folgen-den Zeigern aufgefüllt, die den 'System initial Parameter Block' beschreiben:

mp_mfl	freie Speicherliste
mp_mal	Liste vom zugeteilten Speicher
mp_rover	roving Zeiger

und alle drei Zeiger zeigen auf eine Struktur, die so aufgebaut ist:

m_link	Zeiger auf nächste Struktur oder auf keine, wenn NULL
m_start	Startadresse des Blocks
m_length	Länge des Blocks
m_own	Benutzerprozeßbeschreibung

Eingabeparameter:

long p_mpb Zeiger auf zwölf freie Bytes

Ausgabeparameter: keine

```
$01 = 1          status = Bconstat(dev)
word             status,          dev;
```

Diese Funktion liefert den Status des angesprochenen Gerätes zurück und sagt damit, ob Zeichen empfangen werden können.

Sie können folgende Geräte abfragen: 1=AUX, 2=CON, 3=MIDI

Rückgabewert	Bedeutung
$0000 = 0	Kein Zeichen erhältlich
$FFFF = -1	Nur ein Zeichen erhältlich
ungleich NULL	Es sind Zeichen erhältlich

Eingabeparameter:

word dev Gerätenummer

Ausgabeparameter:

word status Status des Gerätes

```
$02 = 2          zeichen = Bconin(dev)
long             zeichen;
word                          dev;
```

Mit dieser Funktion können Sie ein Zeichen von den Eingabegeräten abholen. Es wird solange gewartet, bis ein Zeichen erhältlich ist. Die oberen 16 Bit sind NULL. Nur die Tastatur (»CON:) stellt eine Ausnahme dar. Wenn in der Systemvariable conterm ($484) Bit 3 gesetzt ist, wird der 'Scancode' in den oberen 16 Bit zurückgegeben.

Es sind folgende Geräte erlaubt: 0=PRT, 1=AUX, 2=CON, 3=MIDI

Eingabeparameter:

word dev Gerätenummer

Ausgabeparameter:

long zeichen Zeichen mit »Scancode

$03 = 3 Bconout(dev, c)
word dev, c;

Sie können hiermit ein Zeichen an Ausgabegeräte schicken. Die Funktion wartet solange, bis das Ausgabegerät das Zeichen abgeholt hat.

Es sind folgende Geräte erlaubt: 0=PRT, 1=AUX, 2=CON, 3=MIDI, 4=KBD, 5=SCR
Wobei 'SCR' alle Zeichen ausgeben kann. Es werden keine Steuerzeichen für den VT 52 Emulator berücksichtigt und für die Zeichen LF, FF, ESC und CR werden die entsprechenden Zeichen aus der Zeichentabelle geholt.

Eingabeparameter:

word dev Gerätenummer
word c Das Zeichen, das gesendet wird

Ausgabeparameter: keine

$04 = 4 error = Rwabs(rwflag, buf, count, recno, dev)
word error, rwflag, count, recno, dev;
long buf;

Diese Funktion erlaubt, daß laden einzelner Sektoren von Diskettenlaufwerken, Festplattenlaufwerken, Ramdisks usw.

Mit der Variable 'rwflag' können Sie folgendes angeben:

rwflag	Bedeutung
0	Lesen
1	Schreiben
2	Lesen, ohne »Media-Change Erkennung
3	Schreiben, ohne »Media-Change Erkennung

'buf' zeigt auf eine Puffer, der die gelesenen oder zu schreibenen Informationen aufnimmt. Der Puffer darf auch auf ungeraden Adressen liegen. Das hat aber zur Folge, daß die Übertragung langsamer wird.

'count' gibt die Menge der zu transferierenden »Sektoren an.

'recno' ist die logische Sektornummer, mit der der Start des Transfers angegeben wird.

Mit 'dev' kann man folgende Geräte ansprechen:

dev	Gerät
0	Diskettenstation 'A:'
1	Diskettenstation 'B:', bei einem System mit einem Laufwerk wird die Diskettenstation 'A:' angesprochen
≥2	Festplattenlaufwerke, Ramdisk, Netzwerke, etc.

Eingabeparameter:

```
word rwflag      gibt den Modus an
long buf         zeigt auf einen Puffer
word count       Menge der Sektoren
word recno       Sektornummer
word dev         Gerätenummer
```

Ausgabeparameter:

```
word error      Fehler bzw. bei erfolgreichen Operationen NULL
```

```
$05 = 5        oldvec = Setexc(vecnum, vec)
word                          vecnum;
long                                  vec;
```

Mit *Setexc()* können Sie einzelne Systemvektoren verändern

'vecnum' gibt die Vektornummer an.

'vec' gibt die Adresse an, auf die der »Vektor zeigen soll, bzw. wenn 'vec' gleich -1 ist, wird nur die Adresse, auf die der Vektor zeigt, zurückgegeben.

Die »Vektoren $00 bis $FF sind für den »M68000 reserviert.

Die »Vektoren $100 bis $1FF sind für »Gemdos reserviert.

Die »Vektoren $200 bis $FFFF sind für OEM-Benutzung reserviert. Diese »Vektoren sind beim Atari ST mit Systemvariablen und Puffern belegt und bei gebooteten »TOS mit dem TOS selber.

Eingabeparameter:

word venum Vektornummer
long vec Neue Adresse des Vektor's bzw. minus eins

Ausgabeparameter:

long oldvec alter Inhalt des Vektor's

$06 = 6 millisec = Tickcal()
long millisec;

Liefert die Zeit zwischen zwei Systemtimeraufrufen in Millisekunden zurück. Das Ergebnis der Funktion ist normalerweise 20. Der Systemtimer ist der Interrupt 5 vom MFP 68901.

Eingabeparameter: keine

Ausgabeparameter:

long millisec Die Millisekunden zwischen zwei Systemtimeraufrufen

---

$07 = 7 zeiger = Getbpb(dev)
long zeiger;
word dev;

'dev' gibt die Gerätenummer des Gerätes an, von welchem der 'Bios Parameter Block' geholt werden soll. Die Geräte haben die Nummern wie in *Rwabs()* beschrieben. Die Funktion liefert einen Pointer auf den »Bios Parameter Block. Wenn die Funktion eine NULL zurückliefert, ist ein Fehler aufgetreten.

Eingabeparameter:

word dev Gerätenummer

Ausgabeparameter:

long zeiger Zeigt auf den »Bios Parameter Block

```
$08 = 8          status = Bcostat(dev)
long             status;
word                            dev;
```

Mit dieser Funktion können Sie feststellen, ob ein Ausgabegerät Sendebereit ist.

Die Funktion liefert -1 zurück, wenn das Ausgabegerät sendebreit ist, und NULL, wenn das Ausgabegeräte nicht sendebreit ist.

Folgende Geräte sind zugelassen: 0=PRT, 1=AUX, 2=CON, 3=MIDI, 4=KBD

Eingabeparameter:

word dev Gerätenummer

Ausgabeparameter:

long status Gerätestatus

```
$09 = 9          status = Mediach(dev)
long             status;
word                            dev;
```

'dev' gibt die gleichen Gerätenummer wie bei *Rwabs()* an. Mit diesen Funktion können Sie feststellen, ob z.B. die Diskette gewechselt wurde (»Media-Change).

Die Funktion kann folgendes zurückliefern:

Wert	Bedeutung
1	Es ist definitiv kein »Media-Change aufgetreten
2	»Media-Change ist vielleicht aufgetreten
3	»Media-Change ist definitiv aufgetreten

Gemdos bemerkt bei Leseoperationen nicht, ob »Media-Change vorliegt. Wenn das Bios bemerkt, daß Media-Change aufgetreten ist, gibt es einen Fehler zurück.

Eingabeparameter:

word dev Gerätenummer

Ausgabeparameter:

long status »Media-Change Status

$0A = 10 **drvbits = Drvmap()**
long drvbits;

Gibt ein Langwort zurück, in dem mit den einzelnen Bit's beschrieben wird, welche Laufwerke angeschlossen sind. Wenn ein Bit gesetzt ist, ist ein Laufwerk angeschlossen. Das Bit Null entspricht dem Laufwerk A: und das Bit 1 B: usw. Die Funktion liefert zurück, was in der »Systemvariablen *drvbits* ($4C4) steht.

Eingabeparameter: keine

Ausgabeparameter:

long drvbits enthält die angeschlossenen Laufwerke

$0B = 11 **oldmode = Kbshift(mode)**
word oldmode, mode;

Hiermit können Sie die verschieden 'Shift Bits' einstellen, wenn 'mode' negativ ist, werden nur die alten 'Shift Bits' zurückgegeben.

Die Bits der Variable 'mode' haben folgende Bedeutung:

Bit	Bedeutung
0	Rechte Shifttaste
1	Linke Shifttaste
2	Controltaste
3	Alternatetaste
4	Chaps-Lock-Taste
5	Rechte Maustaste (ALT CLR/HOME)
6	Linke Maustaste (ALT INSERT)
7	Reserviert

Eingabeparameter:

word mode enthält 'Shift Bits' oder einen negativen Wert.

Ausgabeparameter:

long oldmode enthält die alten 'Shift Bits'

BIOS Parameterblock

Die »BIOS-Routine 7 *Getbpb* liefert einen Zeiger auf den B., der unabhängig von der Art des Laufwerkes (Festplatte oder Floppy) folgenden Aufbau hat:

$00 = 0	sect_siz	Größe eines Sektors in Bytes.
$02 = 2	clst_sec	Größe eines Clusters in Sektoren.
$04 = 4	clst_siz	Größe eines Clusters in Bytes.
$06 = 6	dir_sec	Größe des Directories in Sektoren.
$08 = 8	fat_siz	Größe der »FAT in Sektoren.
$0A = 10	fat_rec	Sektornummer des zweiten »FAT.
$0C = 12	dat_rec	Sektornummer des ersten Datenclusters.
$0E = 14	clst_num	Anzahl aller Datencluster dieses Laufwerks.
$10 = 16	bpb_flag	Diverse Flags

BITBLK

B. ist der Name einer Struktur, die zur Darstellung von »Objekten vom Typ "G_IMAGE" verwendet wird. B. beschreibt ein »Bit-Image näher, welches mit »*objc_draw()* gezeichnet werden kann. Weitere Informationen stehen unter dem Stichwort »Objekt.

Bit Image

Engl.: Bit-orientierte Bilder. B. sind Bilder, bei denen ein gesetztes »Pixel durch ein gesetztes Bit dargestellt wird, so wie das der »Video-Speicher des Atari umsetzt.

Bitblock-Transfer

Siehe »Raster und »Line A

Bootsektor

Der Bootsektor enthält die Informationen, wie eine Diskette aufgebaut ist. Bootsektoren, die es ermöglichen ein TOS von Diskette zu booten, enthalten ein Programm. Die Disketteninformationen sind alle im »Intel-Format abgelegt. Der Bootsektor wird ausge-

führt, wenn die Schecksumme $1234 entspricht. Beim ermitteln der Schecksumme wird nur mit Wörtern gearbeitet.

Wichtige Werte im Bootsektor:

Offset	Bedeutung (alle Werte im »Intel-Format)
$08 = 8	24 Bit lange Seriennummer
$0B = 11	Menge der Bytes eines Sektor
$0D = 13	Anzahl der Sektoren pro »Cluster
$0E = 14	Menge der reservierten Sektoren
$10 = 16	Anzahl der »FAT's
$11 = 17	Menge der möglichen Directoryeinträge
$13 = 19	Anzahl der Sektoren auf Diskette
$15 = 21	Media-Descriptor
$16 = 22	Anzahl der »FAT's auf Diskette
$18 = 24	Zahl der Sektoren pro Track
$1A = 26	Zahl der Seiten auf Diskette
$1C = 28	Anzahl der verborgenden Sektoren

Bootsektor:

```
0000        BRA     Start           * Start anspringen
0002        DC.B    'Loader'        *
0008        DC.B    $EF,$30,$8A     * Seriennummer
000B        DC.W    $0002           * Bytes der Sektoren
000D        DC.B    $02             * Sektoren pro Cluster
000E        DC.W    $0100           * reservierte Sektoren
0010        DC.B    $02             * Menge der FAT's
0011        DC.W    $7000           * Zahl der Directory-
                                    * einträge
0013        DC.W    $A005           * Zahl der Sektoren auf
                                    * Diskette
0015        DC.B    $F8             * Media-Descriptor
0016        DC.W    $0500           * Zahl der Sektoren pro
                                    * FAT
0018        DC.W    $0900           * Zahl der Sektoren pro
                                    * Track
001A        DC.W    $0200           * Zahl der Seiten auf
                                    * Diskette
001C        DC.W    $0000           * Zahl der verborgenden
                                    * Sektoren
001E        DC.W    $0000           * COMMAND laden ? 0 = nein
0020        DC.W    $0000           * Lademodus
                                    * 0 = Datei laden
                                    * <> 0 = Sektoren laden
0022        DC.W    $0000           * erster Sektor
0024        DC.W    $0000           * Menge der Sektoren
```

```
0026          DC.L    $00040000          * Ladeadresse des
                                         * Betriebssystems
002A          DC.L    $00008000          * Adresse des Puffers für
                                         * die FAT
002E          DC.B    'TOS     IMG',$00  * Name des Betrieb-
                                         * systems
003A Start:   MOVE.W  $1E(PC),$000482    * Lade kein COMMAND.PRG
                                         * nach dem booten
0042          MOVE.W  $000446,-(A7)      * Hole Laufwerksnummer
                                         * vom Bootlaufwerk
0048          MOVE.W  #$0007,-(A7)       * Funktionsnummer von
                                         * Getbpb()
004C          TRAP    #13                * Bios aufrufen
004E          ADDQ.W  #4,A7              * Stack reparieren
0050          TST.L   D0                 * Wenn mißglückt
0052          BEQ     Lösche             * abbrechen
0056          MOVE.L  D0,A5              * lade Beginn vom BPB nach
                                         * A5
0058          LEA     $2A(PC),A0         * lade Begin des Puffers
005C          TST.L   (A0)               * nach A0 und teste,ob der
                                         * Wert ungleich Null ist
005E          BNE     $68                * Wenn ja, dann nehme das
                                         * als Puffer
0062          MOVE.L  $000432,(A0)       * andernfalls nehme den
                                         * Wert, den das Boot-ROM
                                         * festgelegt hat
0068          MOVE.W  $0008(A5),D0       * hole FAT-Länge
006C          LSL.W   #8,D0              * multipliziere sie mal 8
006E          ADD.L   D0,D0              * und verdoppel sie
0070          MOVE.W  D0,A4              * lade Länge nach A4
0072          ADDA.L  $2A(PC),A4         *
0076          MOVE.W  $20(PC),D0         * Lademodus
007A          BEQ     $8C                * wenn null, eine Datei
                                         * laden
                                         * wenn nicht, Sektoren
                                         * laden
007C          MOVE.W  $22(PC),D6         * erster Sektor
0080          MOVE.W  $24(PC),D4         * Menge der Sektoren
0084          MOVE.L  $26(PC),A3         * Ladeadresse
0088          BRA     Letzter            *
008C          MOVE.W  $000A(A5),D6       * Startsektor der 2. FAT
0090          MOVE.W  $0008(A5),D4       * FAT-Länge in Sektoren
0094          ADD.W   $0006(A5),D4       * Länge des Directory in
                                         * Sektoren
0098          MOVE.L  $2A(PC),A3         * Beginn des Ladepuffers
                                         * nach A3 laden
009C          BSR     Sektor             * lade Sektoren
00A0          BNE     Lösche             * falls Fehler, Bootvor-
                                         * gang abbrechen
```

```
00A4          MOVE.L  A4,A0            * Lade die FAT-Länge nach
                                       * A0
00A6          MOVE.W  $0006(A5),D0     * lade Directorylänge nach
00AA          LSL.W   #8,D0            * D0 und multipliziere sie
00AC          LSL.W   #1,D0            * mit 256 und mit 2
00AE          LEA     $00(A0,D0.W),A0  * Lade Ende des Directory
                                       * nach A0
00B2          LEA     $2E(PC),A1       * Lade nach A1 die Adresse
                                       * des Namen des Betriebs-
                                       * systems
00B6 Next:    SUBA.W  #$0020,A0        * Ziehe 32 ab für den
                                       * Directoryeintrag
00BA          CMPA.L  A4,A0            * Wenn Directoryanfang
00BC          BLT     Lösche           * Name des Betriebssystems
                                       * nicht gefunden
00C0          MOVEQ   #$0A,D0          * Länge das Namen minus 1
00C2 Gleich:  MOVE.B  $00(A0,D0.W),D1  * Lade Directoryeintrag
                                       * nach D1
00C6          CMP.B   $00(A1,D0.W),D1  * Vergleiche mit dem Namen
                                       * des Betriebssystems
00CA          BNE     Next             * Falls nicht gleich mit
                                       * dem Namen nächsten Ein-
                                       * trags versuchen
00CC          DBF     D0,Gleich        * vergleichen bis D0 -1
                                       * ist
00CE          MOVEQ   #$00,D7          * Lösche D7
00D0          MOVE.B  $001B(A0),D7     * Startcluster nach
00D4          LSL.W   #8,D7            * D7 laden
00D6          MOVE.B  $001A(A0),D7     *
00DA          MOVE.L  $2A(PC),A6       * lade Begin der TPA
                                       * nach A6
00DE          MOVE.L  $26(PC),A3       * Ladeadresse des Be-
                                       * triebssystems
00E2          CLR.L   D4               * Lösche D4
00E4 Loop:    CMP.W   #$0FF0,D7        * Testen auf letzten
00E8          BGE     Letzter          * Cluster
00EA          MOVE.W  D7,D3            * lade den ersten Cluster
00EC          SUBQ.W  #2,D3            * nach D3 und ziehe 2 ab
00EE          MULU.W  $0002(A5),D3     * multipliziere mit Größe
                                       * der Cluster in Sektoren
00F2          ADD.W   $000C(A5),D3     * addiere den Startsektor
                                       * des ersten Cluster dazu
00F6          CMP.W   #$0040,D4        * Wenn größer $40 lade
                                       * Sektor
00FA          BGE     Lade             *
00FC          TST.W   D4               * Wenn Null berechne
00FE          BEQ     rechne           * ersten Sektor
0100          CMP.W   D5,D3            * Sonst nächsten
0102          BEQ     $114             * Sektor
```

```
0104 Lade:    BSR      Sektor            * Lade Cluster
0106          BNE      Lösche            * breche Bootvor-
                                         * gang ab
0108          LSL.L    #8,D4             * erhöhe A3 um
010A          LSL.L    #1,D4             * Clusterlänge
010C          ADDA.L   D4,A3             *
010E rechne:  MOVE.W   D3,D6             * Lade D3 nach D6
0110          MOVE.W   D3,D5             * und D5
0112          CLR.L    D4                * lösche D4
0114          ADD.W    $0002(A5),D4      * Addiere Clustergröße
0118          ADD.W    $0002(A5),D5      * zu D4 und D5
011C          MOVE.W   D7,D2             * lade D7 nach D2
011E          LSR.W    #1,D2             * multipliziere D2 mit 2
0120          ADD.W    D7,D2             * addiere D2 mit D7
0122          MOVE.B   $01(A6,D2.W),D1   * lade Clusternummer nach
0126          LSL.W    #8,D1             * D1 und shifte 8 nach
                                         * links
0128          MOVE.B   $00(A6,D2.W),D1   * lade nach den unteren 8
                                         * Bit von D1
012C          BTST     #0,D7             * Falls Bit Null von D7 =
0130          BEQ      ganz              * Null shifte nicht 4 Bits
0132          LSR.W    #4,D1             * nach rechts
0134 ganz:    AND.W    #$0FFF,D1         * lösche die vier oberen
                                         * Bits
0138          MOVE.W   D1,D7             *
013A          BRA      Loop              * weiter laden
013C Letzter:TST.W     D4                * Falls D4 = 0, keine
013E          BEQ      Ende              * weiteren Sektoren laden
0140          BSR      Sektor            * Sektor laden
0142          BNE      Lösche            * falls Fehler, Bootvor-
                                         * gang abbrechen
0144 Ende:    MOVE.L   $26(PC),-(A7)     * lege auf dem Stack die
                                         * Ladeadresse des Betrieb-
                                         * systems ab
0148          RTS                        * erfolgreich den Bootvor-
                                         * gang beenden
014A Lösche:  CLR.L    D0                * Lösche D0 und breche
014C          RTS                        * den Bootvorgang ab.
014E Sektor:  MOVE.W   $000446,-(A7)     * hole Laufwerksnummer
0154          MOVE.W   D6,-(A7)          * Sektornummer
0156          MOVE.W   D4,-(A7)          * Anzahl der Sektoren
0158          MOVE.L   A3,-(A7)          * Puffer
015A          CLR.W    -(A7)             * Sektor lesen
015C          MOVE.W   #$0004,-(A7)      * Funktionsnummer von
                                         * Rwabs()
0160          TRAP     #13               * Bios aufrufen
0162          ADDA.W   #$000E,A7         * Stack reparieren
0166          TST.W    D0                * auf Fehler testen
0168          RTS                        *
```

```
016A        DC.B 'Neuter Booter',$0D,$0A * komischer Name
0179        DC.B '(C)1985 Atari Corp.',$0D,$0A
018E        DC.W  $0000                  * Ataris Copyrights
0190        DC.W  $0000,$0000            * es folgen
0194        DC.W  $0000,$0000            * unbedeutende Daten
0198        DC.W  $0000,$0000
019C        DC.W  $0000,$0000
01A0        DC.W  $0000,$0000
01A4        DC.W  $0000,$0000
01A8        DC.W  $0000,$0000
01AC        DC.W  $0000,$0000
01B0        DC.W  $0000,$0000
01B4        DC.W  $0000,$0000
01B8        DC.W  $0000,$0000
01BC        DC.W  $0000,$0000
01C0        DC.W  $0000,$0000
01C4        DC.W  $0000,$0000
01C8        DC.W  $0000,$0000
01CC        DC.W  $0000,$0000
01D0        DC.W  $0000,$0000
01D4        DC.W  $0000,$0000
01D8        DC.W  $0000,$0000
01DC        DC.W  $0000,$0000
01E0        DC.W  $0000,$0000
01E4        DC.W  $0000,$0000
01E8        DC.W  $0000,$0000
01EC        DC.W  $0000,$0000
01F0        DC.W  $0000,$0000
01F4        DC.W  $0000,$0000
01F8        DC.W  $0000,$0000
01FC        DC.W  $0000,$4B75            * Checksumme so korrigie-
                                         * en, daß sie $1234 ergibt
```

Bugs

Wie jedes andere Betriebssystem besitzt das TOS leider auch Fehler. Uns erscheint das deutsche Betriebssystem vom November 1985 noch das fehlerfreiste zu sein. Es besitzt leider einen Nachteil: es befindet sich nicht auf ROM's und es ist auch ein bißchen zulang dafür. Ein auffälliger Unterschied zwischen den beiden Betriebssystemversionen ist, das die deutschen Texte gekürzt (verunstalltet) wurden. Ein weiterer tödlicher Unterschied ist (da hat wohl jemand zuviel optimiert), wenn man eine null Byte lange Datei, vom »Desktop aus doppelklickt und sie sich auf dem Bildschirm anzeigen läßt so erscheint entweder garnichts oder 'Quatsch', d.h. man muß den Resetknopf drücken, weil man mit 99 prozentiger Wahrscheinlichkeit daraus nicht mehr zurückkommt.

Alte TOS-Versionen, d.h. von vor November 1985, haben noch einen tödlichen Fehler: man kann nicht auf Dateien zugreifen, die sich in Ordnern mit acht Zeichen langen Namen befinden.

Ein weiterer Fehler ist allen TOS-Versionen gemeinsam: wenn man sehr volle Inhaltsverzeichnisse auf einer Diskette hat, wird beim dritten und vierten Directorywindow nicht mehr alles dargestellt, so daß man meinen könnte, die Diskette sei ziemlich leer.

Mit den folgenden Betriebssystemfehler können Sie ganz leicht eine Datei zerstören: Wenn Sie eine Datei auf sich selber kopieren, bleibt sie nur erhalten, wenn das TOS sie vollkommen in den seinen Kopierpuffer laden kann. An dieser Stelle sollte das TOS diesen Vorgang verbieten, wie es auch bei MS-DOS getan wird.

Dies waren bisher nur Fehler, die im Desktop auffallen. Das TOS besitzt auch Fehler, die für Programmierer sehr ärgerlich sind:

1.) Bei Eingabefeldern in Dialogboxen kann man einstellen, welche Zeichen zugelassen sind. Wenn man nur Großbuchstaben zuläßt, stürzt der Rechner beim Drücken von folgendem Zeichen ab: '_'. Sie können sich durch ein Umprogrammieren der Tastaturtabelle helfen, in dem Sie der Taste z.B. als ASCII-Code ein Leerzeichen (32) zuordnen (so verfahren die Autoren).

2.) Bei Überlauf der Tastaturpuffers können merkwürdige Effekte auftreten. Wenn Sie die Tastatur mit Biosfunktionen abfragen, kann es sein, daß irgendwelche Zeichen auf dem Bildschirm ausgegeben werden. Die Abfrage mit Gemdos erweist sich in diesem Falle als tödlich, denn der Rechner stürzt mit drei Bomben ab. Dies werden auch viele Benutzer von Textverarbeitungsprogrammen bestätigen können. Eine Möglichkeit, diesem Fehler auszuweichen, ist den Head- und Tail-Index des Tastaturpuffer zu manipulieren, so daß er gleich ist. Oder Sie können den Tastaturpuffer vergrößern, so daß es zu keinem Überlauf kommen kann.

3.) Bei Aufruf des TRAP #2 stürzt der Rechner ab, wenn nicht der richtige Wert im Register D0 steht, da er nicht auf einen RTE aufläuft, sondern auf einen RTS und damit die falsche Rückkehradresse vom Stack holt.

4.) Bei Benutzung der Dateioperationen des »Gemdos, bemerkt das Betriebssystem nicht, wenn eine Diskette voll ist. Das muß vom Programmierer vorher getestet werden.

5.) Wenn man ein ROM-Modul als Laufwerk c: benutzt, kann man von dort aus nur 19 mal ein Programm starten (danach stürzt der Rechner ab). Die Lösung ist, das ROM-Modul als ein anderes Laufwerk anzumelden.

6.) In der Verwaltung des »Root-Directory ist auch ein Fehler, der Puffer des Directory scheint zu klein zu sein, so daß das Betriebssystem manchmal nicht merkt, das eine Datei schon existiert und sie nochmal erzeugt, statt sie zu überschreiben. Sie können die neuste Datei mit eine bißchen Glück zurückerhalten, wenn sie alle alten Dateien löschen (auf Datum und Uhrzeit achten). Sie können diesen Fehler umgehen, wenn sie ihre Dateien in Ordner ablegen, da diese als Datei verwaltet werden.

Wir möchten noch auf einen Fehler in den Atari-Handbüchern hinweisen: beim ROM-Port, der Midischnittstelle und dem Energieversorgungsanschluß wurden die Anschlüsse falsch beschrieben. Richtig beschrieben finden sie die »Schnittstellen im Anhang C.

Bus-Error

Ein B. ist zunächst einmal ein Signal, das an die CPU »MC 68000 angelegt werden kann. Wird diese Signalleitung von einem anderen Chip auf Masse gezogen, dann löst das in der CPU eine »Exception aus. Ein B. und ein Adress-Error bewirken jedoch, daß im Gegensatz zu anderen »Exceptions mehr Informationen auf den Stack gelegt werden. Zuerst werden wie üblich der Programmzähler und das »Statusregister auf den Stack gebracht. Danach folgt das Instruction Register. Dieses enthält den gerade von der CPU bearbeiteten »Opcode. Dann kommt ein Langwort, das die *Access Address* angibt. Dabei handelt es sich um die Adresse, auf die die CPU gerade zugreifen will. Das letzte Wort, das auf den Stack gebracht wird, ist das sogenannte Super-Statuswort. Von diesem werden jedoch bisher nur die Bits 0 bis 4 benutzt. Die Bits 0-2 enthalten die »Function-Code Bits der CPU. Bit drei gibt an, ob es sich bei dem Zugriff um einen Lese- (1) oder um einen Schreibzugriff (0) handelt. Bit vier besagt, ob es sich um einen Befehl (0) handelt oder auch nicht (1).

Bei einer »Exception, die aufgrund eines Fehlers auftrat, wird von der entsprechenden Routine im Atari ST ein Speicherbereich mit Prozessordaten gefüllt, der bei einem Reset unangetastet bleibt. Dadurch ist es möglich, den Fehler der zum Absturz geführt hat, weitestgehend zu rekonstruieren. Es werden folgende Daten abgelegt:

$000380	enthält den Wert $12345678, wenn die folgenden Daten gültig sind, d.h. wenn wirklich ein Bus-Error auftrat.
$000384	bis $0003A3 enthält die Datenregister D0 bis D7 in dieser Reihenfolge.
$0003A4	bis $0003BF enthält die Adreßregister von A0 bis A6.
$0003C0	enthält den »Supervisor Stackpointer.
$0003C4	Enthält die Exceptionnummer
$0003C8	Enthält den »User Stackpointer.
$0003CC	bis $0003EB enthält 16 Worte vom Stack.

Button

engl., zu deutsch "Knopf". Ein B. ist ein »Objekt, das der Benutzer durch Anklicken wählen kann. Verwendet werden B.'s in »Dialogboxen und »Alarmboxen, um verschiedene Wahlmöglichkeiten anzubieten. »Objekte vom Typ "G_BUTTON" sind unter »Objekt näher erläutert. B.'s bieten eine bequeme Möglichkeit, eine Auswahl unter verschiedenen Möglichkeiten zu treffen.

C

C ist der Name einer von Dennis M. Ritchie entwickelten Programmiersprache, die etwa zeitgleich mit Pascal erfunden wurde. Auf dem Atari ist C deshalb die Programmiersprache erster Wahl, weil das »Betriebssystem des Rechners selbst überwiegend in C geschrieben worden ist. Daher gestaltet sich die Kommunikation eines Programmes mit dem »Betriebssystem auf eine einfachere Weise als mit anderen Sprachen. C ist sozusagen die **Muttersprache** des Atari.

Capslock

Als Capslock-Taste bezeichnet man eine Taste auf der Tastatur, die alle Buchstaben (und nur Buchstaben!) "shifted" (Capslock = Capital Shift Lock). Auf der Atari-Tastatur finden Sie sie rechts unten neben der Leertaste.

Centronics-Schnittstelle

Hierbei handelt es sich um eine genormte Schnittstelle, die im allgemeinen zur Ansteuerung von Druckern benutzt wird. Der Atari ST besitzt ebenfalls eine C.-S., die jedoch leider nicht mit einer üblichen Centronics-Buchse sondern mit einem 25 poligen Cannon-Stecker ausgestattet ist. Trotzdem erfüllt auch diese annähernd die volle Centronics Funktionalität. An den Pins 2 bis 9 liegen die Daten an. Durch einen Low-Wert am Strobe-Pin wird dem angeschlossenen Gerät gezeigt, daß diese Daten gültig sind und übernommen werden sollen. Pin 11 (Busy) ist ein Eingangspin, der dem Rechner durch einen Highwert anzeigt, daß der Drucker momentan nicht in der Lage ist, Daten entgegenzunehmen. Dies liegt im allgemeinen daran, daß entweder gar kein Drucker angeschlossen ist (dieser Eingang ist über einen 1kΩ Pull-up Widerstand an +5V gelegt) oder daß sein Empfangspuffer voll ist. Eine komplette C.-S. bietet außerdem noch eine Leitung an, die dem Rechner signalisiert, daß der Drucker kein Papier mehr besitzt. Diese Leitung existiert beim Atari ST leider nicht. Das Betriebssystem stellt Routinen zur Verfügung, die diese Schnittstelle unterstützen. Der Gerätestatus kann mit der »GEMDOS Routine $11 *Cprnos()* oder mit der »BIOS Routine $08 *Bcostat()* abgefragt werden. Eine Ausgabe erfolgt mit der »GEMDOS Routine $05 *Cprnout()* oder mit der »BIOS Routine $03 *Bconout()*. Zusätzlich kann mit der »BIOS Routine $01 *Bconstat()* auch der Eingabestatus dieser Schnittstelle abgefragt werden, und mit der »BIOS Routine $02 *Bconin()* kann ein Zeichen von dieser Schnittstelle eingelesen werden. Um Zeichen einlesen zu können, muß die Richtung der Centronics-Schnittstelle umgedreht werden (»Soundchip).

Clipboard

Engl.: Klemmbrett, bezeichnet eine Datei, in die Daten eingetragen werden, die von anderen Programmen ebenfalls genutzt werden sollen. Siehe Stichwort »Scrap Library. Ein C. dient dazu, Daten zwischen verschiedenen Programmen auszutauschen So kann ein Grafik-Programm mittels C. Daten aus einem Tabellenkalkulationsprogramm übernehmen und diese grafisch umsetzen. Dies wird für eine "integrierte Datenverarbeitung" benötigt.

Clipping

Als C. bezeichnet man die Fähgkeit von Grafik-Ausgaberoutinen, ihre Ausgabe auf einen angegebenen Bereich zu beschränken. Alle Ausgaben, die außerhalb des Bereichs liegen, werden abgeschnitten (to clip = abschneiden).

Ein Beispiel: Nehmen wir an, der C.-Bereich beginnt bei X = 100, Y = 50 und die Breite beträgt 200 Pixel, die Höhe 100 Pixel (Sie können mit der »VDI-Funktion »*vs_clip()* diesen Bereich setzen). Wenn Sie nun versuchen, eine Linie von [0; 70] nach [500; 70] zu zeichnen, so wird tatsächlich eine Linie von [100; 70] nach [300; 70] gezeichnet.

Dies ist nützlich, wenn Sie in »Fenstern zeichnen: C. verhindert, daß Sie versehentlich über den Fensterbereich hinauszeichnen.

Cluster

Ein C. besteht beim Atari normalerweise aus 2 »Sektoren. C. ist die Mindestlänge, die eine Datei hat. Wenn eine z.B. Datei 20 Byte lang wäre, würde sie dennoch einen C. belegen, d.h. 1024 Byte.

Desk-Accessory

Siehe »Accessory

Desktop

Der D. ist das Programm, das nach dem Einschalten des Atari läuft. Vom D. aus können Programme gestartet werden, Dateien umbenannt oder gelöscht werden etc. Er entspricht der Betriebssystem-Kommandoebene auf anderen Rechnern. Nur hat man hier eine grafische Benutzeroberfläche, die mausgesteuert ist.

Der D. symbolisiert einen Schreibtisch, auf dem sich Karteikästen mit Daten (= Diskettenstationen mit Dateien) befinden, ein Papierkorb für nicht mehr benötigte Informationen, verschiedene Papiere mit Informationen (= »Fenster). Angewählt werden Funktionen, in dem man mit der »Maus auf etwas zeigt und auf die linke Maustaste drückt ("Anklicken").

Wie der D. genau bedient wird, kann in der zum Atari mitgelieferten Anleitung nachgelesen werden. Obwohl die Bedienung sehr einfach ist, empfiehlt es sich, diese von vorne nach hinten durchzulesen. Es gibt nämlich Feinheiten, die durch Ausprobieren alleine nicht herauszufinden sind...

Im Prinzip ist der D. ein Programm wie jedes andere auch. Was dort gemacht wird, ist allen Programmierern über Betriebssystemfunktionen zugänglich. Der D. macht ausgiebig Gebrauch von »AES und »VDI. Er ist damit eine besondere Beispielanwendung, da er jedem Benutzer des Atari zur Verfügung steht. Von dort werden alle weiteren »Anwendungen aufgerufen, und in der Regel findet man sich nach Programmende auf dem D. wieder.

DESKTOP.INF

Nach dem Einschalten oder nach einem Reset sucht das Betriebssystem des Atari ST zuerst auf dem Laufwerk C, dann auf dem Laufwerk A nach einem File mit dem Namen "DESKTOP.INF". Wird es gefunden, dann wird es geladen und ausgewertet. Dieses File enthält etliche Parameter zum Einstellen des »Desktops und der diversen Schnittstellen. Alle Angaben sind im »ASCII-Format abgelegt. Dadurch ist es möglich, die Parameter mit einem Editor zu verändern. Dieses File kann auch vom »Desktop erzeugt werden, indem Sie in der Menüzeile "Extras" auswählen und dann "Arbeit sichern" anklicken. Die Informationen im File DESKTOP.INF haben folgende Bedeutung. Jede Parameterzeile beginnt mit einem Doppelkreuz, dem ein Buchstabe folgt. Diesem folgen dann weitere Parameter, die meist durch Leerzeichen getrennt sind. Von jedem der folgenden Strings können auch mehrere im File enthalten sein. Der Parameter *Icon* kann folgende Werte annehmen.

00 = Icon für ein Laufwerk.
01 = Icon für ein »Subdirectory.
02 = Icon für einen Mülleimer.
03 = Icon für ein Programm.
04 = Icon für ein normales Datenfile.

#D P_1 Icon Directory
#D FF 01 @ *.*@
Directory. Hiermit wird festgelegt, welches Icon »Subdirectories erhalten sollen. Dabei ist es sogar möglich, für bestimmte Directorynamen verschiedene Icons anzugeben, indem einfach mehrere dieser Strings auftauchen.

#E P_1 Auflösung
#E FB 02
Environment. Legt die Arbeitsumgebung des »Desktops fest. Die Bits des Parameters P_1 haben folgende Bedeutung:
Bit 0-2 werden bisher nicht benutzt.
Bit 3 = Sicherheitsabfrage beim Kopieren: 0=Nein, 1=Ja.
Bit 4 = Sicherheitsabfrage beim Löschen: 0=Nein, 1=Ja.
Bit 5/6 Sortierung erfolgt nach: 00=Name, 01=Datum, 10=Größe oder nach 11=Typ.
Bit 7 = Files als Text (1) oder als Bild (0) anzeigen.
Der Parameter *Auflösung* gibt die Bildschirmauflösung an, mit der gearbeitet werden soll. Dabei ist zu beachten, daß in allen Modi genau dann Monochrome gilt, wenn die entsprechende Leitung am Monitorstecker auf Masse liegt.
01 = 320*200 Punkte in 16 Farben oder Monochrome.
02 = 640*200 Punkte in 4 Farben oder Monochrome.
03 = Monochrome oder 320*200 Punkte.

#F Icon P_2 Programm Datenfile
#F 03 04 *.TOS@ @
TOS-Anwendung. Mit diesem String werden Programmdateien bestimmt, die unter »TOS laufen sollen. Im Beispiel sind dies alle Files, deren Name mit ".TOS" endet. Wird ein *Datenfile* angegeben, dann wird nach einem Doppelklick auf das Datenfile (z.B. Files mit der Endung ".LEX") zuerst das davorstehende Programm geladen, welches dann dieses Datenfile nachlädt. Ein solcher Eintrag könnte z.B. so aussehen:
#F 03 04 TEXTPROG.PRG *.LEX@
Dann führt ein Doppelklick auf eines dieser Lexikon Textfiles dazu, daß zuerst das Textprogramm geladen wird. Die Bedeutung des Parameters P_2 konnte noch nicht geklärt werden.

#G Icon P_2 Programm Datenfile
#G 03 FF *.APP@ @
GEM-Anwendung. Dieser String bestimmt Programme, die »GEM benutzen. Im Beispiel sind dies alle Programme, die die Endung

".APP" besitzen. Der Parameter P_2 ist unbenutzt, hat also keine Bedeutung. Für das Datenfile gilt das bereits für die TOS-Anwendung gesagte.

#M X_Pos Y_Pos Icon P_4 Drive Name
#M 00 00 00 FF A Diskstation@ @
Laufwerk. Gibt die Position des Icons für das Laufwerk an. Der Nullpunkt des Koordinatensystems befindet sich in der linken oberen Ecke. Um die reale Position in Pixeln zu erhalten, muß die X-Koordinate noch mit 80 multipliziert werden. Die Y-Koordinate muß noch mit 40 multipliziert werden, bzw. mit 20, wenn in einem der Farbmodi gearbeitet wird. Zur Y-Position sind noch 20 Pixel hinzuzuaddieren, die für die Menüzeile benötigt werden. Der Parameter P_4 hat scheinbar keine Bedeutung. Für den Parameter *Drive* wird ein Großbuchstabe eingesetzt und der Parameter *Name* darf bis zu 12 Zeichen lang sein.

#P Icon P_2 Programm Datenfile
#P 03 04 *.TTP@ @
TTP-Anwendung. Dieser String legt die Parameter für TTP-Anwendungen fest. Es gilt das bereits unter TOS-Anwendung gesagte.

#T X_Pos Y_Pos Icon P_4 Name
#T 01 00 02 FF Büroablage@ @
Trash (Mülleimer). Die ersten zwei Parameter legen die Position des Mülleimers fest, dann folgt der Icontyp, P_4 ist ohne Bedeutung und der Name darf wie beim Icon für das Laufwerk 12 Zeichen lang sein.

#W X_bar Y_bar X_Pos Y_Pos Breite Höhe P_7 Pfad
#W 00 00 0A 01 1D 18 08 A:*.*@
Windowdefinition. Die Parameter *X_bar* und *Y_bar* legen fest, wie weit die Schieber am unteren bzw. am rechten Ende des Fensters vom Nullpunkt aus entfernt sind. Die Parameter *X_Pos* und *Y_Pos* legen die Koordinaten der linken oberen Ecke des Fensters fest. Diese sind etwas anders kodiert als beim Laufwerk. Um die realen Werte zu erhalten, müssen die Parameter *X_Pos* und *Breite* noch mit 8 multipliziert werden. Die Parameter *Y_Pos* und *Höhe* müssen bei monochromem Bildschirm mit 16, sonst mit 8 multipliziert werden. Der Parameter P_7 ist unklar. Wird bei *Pfad* ein »Pfad angegeben, so versucht das Betriebssystem, das zugehörige Directory einzulesen und anzuzeigen.

#a012345
#a000020
V.24-Einstellung. Dieser String legt die Arbeitsweise der »V.24-Schnittstelle fest. Die sechs Parameter sind alles Zahlen zwischen Null und Neun, bzw. die entsprechenden »ASCII-Zeichen.

Es gilt:
0: 0=Vollduplex, 1=Halbduplex.
1: Baudrate 0=9600, 1=4800, 2=1200, 3=300.
2: Parität 0=keine Paritätserkennung, 1=ungerade, 2=gerade.
3: Bits je Zeichen 0=8, 1=7, 2=6, 3=5.
4: Bit 0 = XON/XOFF aus (0), oder an (1).
Bit 1 = RTS/CTS aus (0), oder an (1).
5: Bit 8 ist gesetzt (0), oder nicht (1). Dies gilt natürlich nur bei weniger als 8 Bit je Zeichen.

#b*012345*
#b001100
Druckeranpassung. Die Parameter 0 bis 5 legen die Arbeitsweise der Druckerschnittstelle fest, wie sie auch mit dem »Accessory "CONTROL.ACC" einstellbar sind. Die Parameter haben folgende Bedeutung:

0: Druckertyp	0=Matrix	1=Typenrad.
1: Farbe	0=Schwarz/Weiß	1=Farbe.
2: Punkte je Zeile	0=1280	1=960.
3: Qualität	0=Draft	1=Near Letter Quality.
4: Schnittstelle	0=Centronics	1=»V.24-Schnittstelle.
5: Papierart	0=endlos	1=Einzelblatt.

#c*777...411*P_1P_2
#c000...3101903
Kontrollfeld. Direkt hinter dem 'c' folgen 48 Zeichen, von denen jeweils drei aufeinanderfolgende einen der 16 Farbwerte festlegen. Jede Ziffer kann einen Wert zwischen 0 und 7 annehmen. Die erste Ziffer ist die Farbintensität für Rot, dann folgt Grün, dann Blau. Diese drei Intensitäten legen fest, wie die Farbe aussieht. Dies läßt sich mit den Rollbalken im Kontrollfeld ebenfalls einstellen. Der Parameter *4* legt die Geschwindigkeit der Maus fest und kann Werte von Null bis vier annehmen. Der nächste Parameter schaltet den Tastaturklick ein (1) oder aus (0). Dann wird die Glocke ein- (1) oder ausgeschaltet (0). Der Parameter P_1 gibt die Zeit an, die vergeht, bevor eine immer noch gedrückte Taste wiederholt wird. Der letzte Parameter P_2 legt die Wiederholgeschwindigkeit fest. Die beiden letzten Parameter bestehen aus jeweils zwei Zeichen.

#d
#d
???. Die Bedeutung dieses Strings ist nicht bekannt. Sichert man seine »Desktop Oberfläche ab, dann stehen im allgemeinen am Anfang des Files die Strings mit den Kleinbuchstaben, dann kommt dieser String und daran schließen sich die Strings mit den Großbuchstaben an. Möglicherweise hat dieser String eine Art Trennfunktion, oder ist für spätere Erweiterungen reserviert.

Dialogbox

Eine D. ist nichts weiter als ein "interaktives Formular", in das Daten in Form von Texten eingetragen werden können und / oder durch Zeigen mit der »Maus zwischen verschiedene Möglichkeiten (»Objekten) gewählt werden kann.

Bevor Sie an dieser Stelle weiterlesen, sollten Sie sich das Stichwort »Objekt **auf jeden Fall** näher angesehen haben! Glauben Sie mir, alles andere führt in die Verzweiflung!

Eine D. hat drei Repräsentationen: die **logische** Struktur, die das Verhältnis der Teile zum Ganzen angibt, die **physikalische** Anordnung, die die Anordnung der Daten im Speicher darstellt, und **visuelle** Darstellung, also daß, was der Benutzer nachher auf dem Schirm sieht. Die physikalische Anordnung ist unter dem Stichwort »Objekt erläutert. Jede D. wird als ein »Objektbaum im Speicher repräsentiert. Dies hört sich zunächst furchtbar kompliziert, aber es ist nicht ganz so schlimm. Immerhin haben Sie sich (in weiser Voraussicht) dieses Buch zugelegt, um nicht in unbekanntem Gelände mit Ihrem Rechner abzustürzen.

Also auf in das (noch) unerforschte Gebiet der Dialogboxen. Zunächst ein Beispiel:

Wenn Sie im Desktop die Menüleiste links oben anwählen und dort den Menüpunkt "Desktop Info" anklicken, so erscheint in der Mitte des Bildschirms ein weißer Kasten mit Umrahmung. In diesem Kasten steht die Copyright-Meldung von Atari und Digital Research sowie (unten) ein "Knopf" (den ich im folgenden immer als "»Button" bezeichnen möchte).

Der weiße Kasten (die Box) ist das sog. "Wurzelobjekt" - was sich daran erkennen läßt, das die Box zuerst gezeichnet wird. In dieser Box befinden sich verschiedene Textzeilen. Diese Textzeilen bezeichnet man als "Abkömmlinge" oder "Kinder". Ferner kann man auch noch einen »Button sehen (unten). Auch dies ist ein Abkömmling des Wurzelobjektes "Box". Jedes Kind kann noch weitere Kinder haben (was in unserem Beispiel nicht der Fall ist). Anders herum gesagt, hat jedes Kind ein Elternteil (nur eines!). Das Elternteil des Wurzelobjekts ist der Bildschirm.

Das Wurzelobjekt ist die oberste Ebene: dieses Objekt kann wieder mehrere Objekte beinhalten (zweite Ebene), die wiederum mehrere Objekte beinhalten können (dritte Ebene) usw. Wobei noch gesagt werden sollte, daß "beinhalten" meint, daß der Abkömmling sich vollkommen innerhalb des Elternobjektes befinden **muß**. Es gibt also keine Überschneidung von Elternteil und Kind. Wenn sich etwas überschneidet, so sind dies Kinder von ein und demselben Elternteil!

Wenn Sie die eben beschriebene Struktur einmal genauer betrachten, so werden Sie feststellen, daß sie einer Baumstruktur ähnlich sieht. Daher also der Ausdruck "Wurzelobjekt"! Und deshalb bezeichnet man das ganze Gebilde auch als "Baum". In dem Beispiel, das unter dem Stichwort Objekt aufgeführt wurde, kann man dies auch annähernd erkennen, obwohl die Wurzel oben ist.

Zusammenfassend läßt sich folgendes sagen: Jedes Elternteil und jedes Kind ist ein »Objekt. Eine Reihe von Objekten wird in einem »Objektbaum zusammengefaßt. Beispielsweise ist eine Dialogbox ein »Objektbaum. Die Daten der Objekte werden in einer Struktur vom Typ "OBJECT" zusammengefaßt. Der »Objektbaum besteht aus einem Array von Strukturen vom Typ "OBJECT". Jedes »Objekt hat eine Indexnummer, über die es angesprochen werden kann.

Wie kommt man an solche Objektstrukturen? Natürlich ist es möglich, diese "zu Fuß" zu programmieren (insbesondere in C). Besser ist es jedoch, mit Hilfe des "Resource Construction Sets" (Abk.: RCS) die Objekte (z. B. Dialogboxen) zu entwerfen.

Die Baumstruktur wird von dem RCS-Programm in ein sog. "RSC-File" geschrieben. Diese Datei wird später von Ihrem Programm nachgeladen. Das geht mit einem Aufruf der Routine *rsrc_gaddr()*:

```
ok = rsrc_load(RSC_Name)
```

Der Parameter 'RSC_Name' ist die Adresse des Namens des »RSC-Files, z. B. "MEINPRG.RSC". Allgemeiner Konvention entsprechend lautet die »Extension des Dateinamens stets auf ".RSC". "MEINPRG" sollte durch den Namen Ihres Programms ersetzt werden. Die Variable 'ok' wird in dem Register D0 zurückgegeben (wie üblich). Sie ist gleich Null, wenn irgendein Fehler aufgetreten ist (z. B. Resource-File nicht vorhanden). Dann sollte das Programm abgebrochen werden!

Nehmen wir einmal an, in dem RSC-File befindet sich eine Dialogbox, der Sie mit dem Resource Construction Set den Namen "DIALOG" gegeben haben. Dann müssen Sie im Programm als nächstes herausfinden, wo sich nach dem Laden Ihre Dialogbox (genauer: die Struktur, die Ihre Dialogbox beschreibt) befindet. Das geht so:

```
                rsrc_gaddr(0, DIALOG, baum_zeiger);
word                    0, DIALOG;
word *                             baum_zeiger;
```

word 0 — Dieser Parameter gibt den gewünschten Typ der Struktur an, deren Adresse geholt werden soll. Unter dem Stichwort »*rsrc_gaddr()* sind die möglichen Typen aufgelistet, unter dem Stichwort »Objekt sind die Typen näher erklärt.

DIALOG — Der Name, den Sie im RCS Ihrer Dialogbox gegeben haben. Das RCS erzeugt mit den symbolischen Namen eine Datei namens "MEINPRG.H". In dieser steht z. B.:

```
#define DIALOG 0
```

In C steht am Anfang des Programms ein

```
#include "MEINPRG.H"
```

In Pascal geht das so aber nicht. Da müssen Sie schon von Hand in rsrc_gaddr statt des Wortes "DIALOG" die Zahl (in unserem Beispiel: 0) eintragen!
Bei dieser Nummer handelt es sich um die **Indexnummer** des »Objekts im »Objektbaum.

baum_zeiger — Ein Zeiger auf eine Variable, in der die Addresse Ihrer Dialogbox stehen soll (**nicht die Variable selbst!**). In C geht das so:

```
OBJECT *baum_zeiger;
```

Damit wird eine Variable definiert (die global sein sollte!), die ein Zeiger auf eine Struktur vom Typ "OBJECT" ist. Nun schreiben Sie oben:
rsrc_gaddr (0, DIALOG, &baum_zeiger);

Nun kann's losgehen - wir produzieren eine Dialogbox! Übrigens: Ich empfehle Ihnen, jede hier verwendete Routine nachzuschlagen. Das kann Ihnen eine Menge Kopfschmerzen ersparen!

Im folgenden beschreibe ich in allgemeiner Form eine Routine, die zur Darstellung und Auswertung einer Dialogbox verwendet werden kann. An diese Routine müssen Sie zwei Parameter übergeben: 'dialog_addr', ein Langwort, das die Adresse der Dialogbox enthält (nach obigem Muster ermittelt) und 'ed_start'. Letzteres ist ein Wort, dessen Bedeutung ich später erkläre. Jetzt nehmen wir erst einmal an, es sei Null. Die Beschreibung der Routine erfolgt analog zu den Erläuterungen der Betriebssystem-Routinen. Die Aufrufe erscheinen in Fettschrift. Was nicht fett gedruckt wurde, ist als Kommentar zu betrachten!

Die Routine beginnt:

```
            ret_val = dialog_handler(dialog_addr, ed_start)
word        ret_val,                                ed_start;
word *                                  dialog_addr;
```

word ret_val — Der Rückgabewert der Routine. Er ist in diesem Fall die Indexnummer des Objekts, welches zur Beendigung des Dialogs vom Benutzer ausgewählt wurde

word *dialog_addr — Adresse des Objektbaums, der die Beschreibung der Dialogbox enthält

word ed_start — Wenn die Dialogbox editierbaren Text enthält, so wird mit 'ed_start' die Indexnummer desjenigen Textfeldes übergeben, in dem zu Beginn des Dialogs der Cursor stehen soll. Sonst eine Null übergeben!

Nun müssen Sie vier Variablen deklarieren und definieren (d. h., für diese Werte muß Speicherplatz zur Verfügung stehen:

word xdial — X-Koordinate, an der die Dialogbox angezeigt werden soll

word ydial — Y-Koordinate, an der die Dialogbox angezeigt werden soll

word wdial — Die Breite der Dialogbox in Pixeln

word hdial — Die Höhe der Dialogbox in Pixeln

Die Werte für diese vier Variablen muß nun berechnet werden, dies geschieht mit der Routine »*form_center()*:

```
            form_center(&dialog_addr, &xdial_addr, &ydial_addr,
word *                  dialog_addr,  xdial_addr,  ydial_addr;
                       &wdial_addr, &hdial_addr)
word *                  wdial_addr,  hdial_addr;
```

word *dialog_addr — Oben erklärt (im weiteren werde ich nur die neuen Variablen darlegen)

word *xdial_addr
word *ydial_addr
word *wdial_addr
word *hdial_addr — Bei diesen vier Zeigern handelt es sich um die Adressen der Variablen 'xdial', 'ydial' usw.

(In C schreibt man dafür '&xdial' etc). Nach Aufruf von »*form_center()* enthalten die Variablen 'xdial' etc. die bei der Beschreibung oben angegebenen Werte.

Bevor wir nun irgend etwas auf dem »Bildschirm unternehmen, schalten wir erst einmal den »Mauszeiger ab. Wenn man über den »Mauszeiger hinweg zeichnet und dieser dann bewegt wird, so wird der **alte** Hintergrund des »Mauszeigers restauriert. Dieses "Loch" sieht nicht gut aus! Deshalb verwenden wir die »VDI-Routine »*v_hide_c()*:

```
            v_hide_c(handle)
word                handle;

word handle     »Handle der »Workstation
```

Nun muß AES über die nahende Aktion benachrichtigt werden, damit wir Platz auf dem Bildschirm bekommen (und somit später dafür gesorgt wird, daß der Desktop-Hintergrund neu gezeichnet wird und an alle Fenster, die von der Dialogbox überschrieben wurden, eine Nachricht zum Neuzeichnen gesandt wird).

```
        form_dial(0, 0, 0, 0, 0, xdial, ydial, wdial, hdial)
word              0, 0, 0, 0, 0, xdial, ydial, wdial, hdial;
```

Bitte bei Bedarf unter »*form_dial()* nachschlagen!

Was jetzt kommt, kann weggelassen werden: Wir zeichnen eine sich vergrößernde Box. Dies sieht hübsch aus, verschwendet aber ansonsten lediglich Zeit (dieser Aufruf ist optional):

```
        form_dial(1, 0, 0, 0, 0, xdial, ydial, wdial, hdial)
word              1, 0, 0, 0, 0, xdial, ydial, wdial, hdial;
```

Als nächstes lassen wir die Dialogbox zeichnen:

```
        objc_draw(dialog_addr, ROOT, MAX_DEPTH,
word *            dialog_addr;
word                         ROOT, MAX_DEPTH;
                  xdial, ydial, wdial, hdial)
word              xdial, ydial, wdial, hdial;

word ROOT       Eine Konstante, die angibt, von welchem Objekt an
                der Objektbaum gezeichnet werden soll. Im allge-
                meinen nimmt man dafür Null, d. h. die Dialogbox
                wird vom Wurzelobjekt an dargestellt.
```

word MAX_DEPTH MAX_DEPTH bezeichnet die maximale "Tiefe", bis zu der gezeichnet werden soll. Wenn unser Baum aus 20 Ebenen besteht (sich also 20 mal verzweigt), und wir geben als Root 5 und als MAX_-DEPTH 15 an, so wird der Baum von der fünften bis zur fünfzehnten Ebene gezeichnet, den Rest läßt »GEM weg. Geben wir als MAX_DEPTH 25 an, so zeichnet »GEM aber nur bis zur letzten vorhandenen Ebene!

An dieser Stelle steht nun Ihre Dialogbox in voller Schönheit auf dem »Bildschirm. Was jetzt noch getan werden muß, ist die Interaktion des Benutzers mit der Dialogbox zu überwachen. Das ist nicht so kompliziert, wie es sich anhört, da es auch dafür eine Funktion gibt, die Sie einfach aufrufen können. Vorher sollten Sie aber noch den Mauscursor wieder einschalten, da es sich mit einem unsichtbaren Zeiger so schlecht zeigen läßt:

```
            v_show_c(handle, reset)
word                 handle, reset;

word handle        »Handle der »Workstation

word reset         In diesem Fall den Wert Null nehmen (siehe
                   »v_show_c())

            ret_value = form_do(dialog_addr, ed_start)
word        ret_value,                       ed_start;
word *                          dialog_addr;
```

In 'ret_value' steht - nachdem die Routine zurückgekehrt ist die Indexnummer des »Objekts, welches der Benutzer zum Beenden des Dialogs angewählt hat (in 'ob_flag' der Objektstruktur dieses »Objekts ist Bit zwei gesetzt worden). Dieses »Objekt bezeichne ich als "EXIT-Button". In 'ob_state' ist jetzt Bit Null gesetzt (=SELECTED), ebenso bei jedem anderen Objekt auch, das der Benutzer gewählt hat. Wenn man beispielsweise herausfinden will, welche der Buttons der Benutzer während des Dialogs gewählt hat (diese Buttons sind schwarz gezeichnet worden), so muß von den entsprechenden Objekten das Bit Null in 'ob_state' getestet werden.

Wenn nun die Dialogbox nochmal dargestellt wird (mittels »*objc_-draw()*), so wird der EXIT-Button invertiert (schwarz) dargestellt: in 'ob_state' ist ja Bit Null (= SELECTED) immer noch gesetzt! Also empfiehlt es sich, dies zu verhindern, indem man das Bit zurücksetzt. Wie dies im einzelnen zu geschehen hat, ist sehr von der verwendeten Programmiersprache abhängig. Die Adresse von 'ob_state' findet man mit folgender Formel:

Adresse von 'ob_state' = 'dialog_addr' + 'ret_val' * 12

'dialog_addr' enthält die Grundadresse der Baumstruktur. Dazu muß die Indexnummer mal 12 addiert werden. Warum mal 12? Weil das Wort 'ob_state' mit dem 12. Byte der OBJECT-Struktur beginnt (siehe »Objekt). In C geht das einfacher:

```
dialog_addr[ret_val].ob_state = ...
```

Damit kann die gewünschte 'ob_state'-Variable angesprochen werden. Um das "SELECTED"-Bit zurücksetzen zu können, schreibt ein C-Programmierer:

```
dialog_addr[ret_val].ob_state &= ~SELECTED;
```

Um zu testen, ob das "SELECTED"-Bit gesetzt ist, schreiben wir:

```
if (dialog_addr[object].ob_state & SELECTED) ...;
```

Wobei 'object' die Indexnummer des zu testenden Objekts ist!

Wenn Ihnen diese gesamte Bit-Rechnerei zuviel ist: das Ganze läßt sich auch durch Aufruf von »*objc_change()* erzielen. Mit der obigen Methode geht es aber eleganter und schneller.

Aber weiter mit unserer Dialog-Routine. Falls Sie optische Effekte lieben, können Sie ein sich verkleinerndes Rechteck zeichnen lassen:

```
form_dial(2, 0, 0, 0, 0, xdial, ydial, wdial, hdial)
```

Nun muß AES nur noch angewiesen werden, für ein Neuzeichnen des Bildschirms zu sorgen. Dies geht so:

```
form_dial(3, 0, 0, 0, 0, xdial, ydial, wdial, hdial)
```

Damit keine Mißverständnisse aufkommen: nur der Desktop-Hintergrund selbst wird von AES neu gezeichnet! Liegen in dem Bereich von 'xdial' etc. Fenster, so generiert AES für diese eine Nachricht (siehe Message). Für den Fensterinhalt ist die »Application zuständig.

Das letzte, was Sie in dieser Routine noch tun sollten, ist die Rückgabe der Variablen 'ret_val' (in C: return(ret_val);), damit die aufrufende Routine weiß, wie das Ganze ausgegangen ist...

Damit ist dann das Ende von 'dialog_handler()' erreicht.

Es gibt aber noch eine Reihe von Fragen: Wie geht man mit editierbaren Textfeldern um?

Das ist verhälnismäßig einfach. Vor den Aufruf von 'dialog_handler() hat »GEM aber noch etwas Schweiß gesetzt. Wir müssen ein paar Vorbereitungen treffen. Nehmen wir mal an, Ihre Dialogbox (deren Addresse Sie mit dem Parameter 'dialog_addr' übergeben haben) enthalte editierbaren Text. Dann steht in 'ed_start' die Nummer des Textfeldes, in dem der Cursor zu Anfang stehen soll. Falls Sie kein editierbares Textfeld haben, so übergeben Sie eine 0. Die Nummer des Textfeldes ist nichts anderes als die Indexnummer desjenigen Objekts, das editiert werden soll ('ob_type' ist "G_FTEXT" oder "G_FBOXTEXT", in 'ob_flags' ist Bit drei gesetzt = EDITABLE).

Nehmen wir des weiteren an, Sie haben in Ihrer Dialogbox mit Hilfe des Resource Construction Sets ein editierbares Textfeld eingebaut und dies mit dem Namen ED_TEXT (= symbolischer Name für die Indexnummer des Objekts) versehen. Dies Textfeld möge genau zur Eingabe von 10 Zeichen geeignet sein. Wie geht das?

Im Resource Construction Set basteln Sie ein entsprechendes Textfeld in Ihre Box ein. Klicken Sie das Feld einmal an, und wählen Sie im Menü die Option "Name" aus. Vergeben Sie den Namen "ED_-TEXT". Nun doppelklicken Sie das Textfeld, und vor Ihren Augen entsteht eine (große) Dialogbox. Ignorieren Sie erst einmal alles, was in dieser Box oben steht, und lenken Sie Ihr Augenmerk nach unten. Dort sehen Sie folgendes:

```
PTMPLT>EDIT:~~~~~~_____________________________

PVALID>~~~~~XXXXXX_____________________________

PTEXT>~~~~~____________________________________
```

Ptmplt ist dabei die Eingabemaske. In der Dialogbox sieht der Benutzer später ja nur:

```
EDIT:______
```

Dort, wo die Schlangenlinie in der Eingabemaske steht ("Tilde") kann später der Benutzer seine Eingabe machen. So wie beschrieben, können nur sechs Zeichen eingeben werden. Erweitern Sie die Anzahl der einzugebenden Zeichen!

Von Pvalid sieht der Benutzer nichts, denn Pvalid gibt an, welche Zeichen bei der Eingabe gültig (englisch: valid) sind. Dabei bedeuten:

9 Bei der Eingabe werden nur Zahlen angenommen (0-9)

A Erlaubt sind nur Großbuchstaben (A-Z)

a Groß- und Kleinbuchstaben (A-Z und a-z)

N Großbuchstaben und Zahlen (A-Z und 0-9)

n Groß- und Kleinbuchstaben sowie Zahlen (A-Z, a-z und 0-9)

F Zugelassen sind Zeichen, die in Dateinamen gültig sind

P Gültig sind Zeichen, die in Datei- und Pfadnamen erlaubt sind, also auch ":" und "\"

p genau wie bei "P" auch

X Erlaubt sind alle Zeichen

Ptext gibt an, welche Zeichen nachher beim Zeichnen der Dialogbox schon im Eingabefeld stehen. In obigem Beispiel bewirkt

PTEXT>~~~~~Hurra!_____

daß der Benutzer bei erstem Aufruf der Dialogbox folgendes zu Gesicht bekommt:

EDIT:HURRA!

Nun kann er das Wort "HURRA!" beliebig editieren.

Im Programm müssen nun verschiedene Werte der zu dem editierbaren Textfeld gehörenden TEDINFO-Struktur mit Werten belegt werden. In unserem Beispiel sollte 'te_ptext' auf den zu editierende String zeigen. Wir erinnern uns: beim editierbaren Textfeld zeigt die Variable 'ob_spec' auf die zugehörige TEDINFO-Struktur. Die Variable 'baum_zeiger' enthält die Adresse der Baumstruktur. Die Adresse der Variablen 'ob_spec' bekommen wir mit folgender Formel:

Adresse von 'ob_spec' = 'baum_zeiger' + 'ED_TEXT' * 12

Die Adresse von 'te_ptext' erhalten wir demnach so:

Adresse von 'te_ptext' = 'ob_spec' + 0

'ob_spec' ist nicht die Adresse von 'ob_spec', sondern der **Inhalt**! "0" ist der der Offset von 'te_ptext' in der TEDINFO-Struktur. In die so gewonnene Adresse schreiben wir die Adresse unse-

res Strings hinein. In diesem String finden wir dann später den Eintrag des Benutzers wieder. Dieselbe Aktion müssen wir noch unternehmen, um die Länge des Strings einzutragen (die Länge des editierbaren Textfeldes war in unserem Beispiel zehn, die Länge des Strings aber ist elf, da wir das abschliessende Nullbyte noch berücksichtigen müssen). Dies wird in 'te_txtlen' eingeschrieben. Die Adresse von 'te_txtlen' erhalten wir so:

Adresse von 'te_txtlen' = 'ob_spec' + 22

22 ist der Offset von 'te_txtlen' in der TEDINFO-Struktur.

Das Ganze sieht sehr kompliziert aus. In C sieht das übrigens so aus:

```
((TEDINFO *)baum_zeiger[ED_TEXT].ob_spec)->te_ptext = string;

((TEDINFO *)baum_zeiger[ED_TEXT].ob_spec)->te_txtlen = 11;

dialog_handler(baum_zeiger, ED_TEXT);
```

(TEDINFO *) ist ein sog. "Cast-Operator", zu deutsch Gipsverband. Damit wird der Compiler gezwungen, von dem nachfolgenden Text anzunehmen, daß es sich um einen Zeiger auf eine TEDINFO-Struktur handelt.

Also bedeutet

```
(TEDINFO *)baum_zeiger[ED_TEXT].ob_spec ...
```

das die Variable 'ob_spec', die in der Struktur 'baum_zeiger[ED_-TEXT]' steckt, auf eine Struktur von obigem Typ zeigt. Wenn "XYZ" ein Zeiger auf eine Struktur ist, so ist XYZ->variable die Komponente "variable" der Struktur "XYZ" selbst. Also ist

```
((TEDINFO *)baum_zeiger[ED_TEXT].ob_spec)->te_ptext
```

die Variable 'te_ptext' in der Tedinfo-Struktur, auf die 'ob_-spec' zeigt. Der Zeiger 'ob_spec' wiederum ist enthalten in der Struktur mit dem Namen 'ED_TEXT' in dem Baum, auf den 'baum_zeiger' zeigt.

Wenn "string" ein Zeiger auf die zu editierende Zeichenkette ist, so wird mit

```
((TEDINFO *)baum_zeiger[ED_TEXT].ob_spec)->te_ptext = string;
```

der Zeiger 'te_ptext' auf diesen String gerichtet. Hinterher steht dann in "string" die Eingabe des Benutzers! Übrigens: Ihr

Gedanke, "te_ptext" habe irgendetwas mit dem "PTEXT" aus dem Resource Construction Program zu tun, ist völlig richtig. "te" steht nämlich für "TEDINFO".

Wenn Sie sich die Tedinfo-Struktur genau ansehen, dann können Sie sehen, daß

```
((TEDINFO *)baum_zeiger[ED_TEXT].ob_spec)->te_txtlen = 11;
```

der Variablen 'te_textlen' die Länge des Strings zuweist (die, wie oben erläutert, in unserem Beispiel elf ist). 'string' muß vorher deklariert worden sein, z. B. als

```
char string[11];
```

Jetzt braucht nur noch die Routine *dialog_handler()* aufgerufen zu werden, was es damit auf sich hat, ist oben beschrieben. Hier nun die Zusammenfassung der oben beschriebenen Routine:

```
dialog_handler(dialog_addr, ed_start)
word *          dialog_addr;
word            ed_start;
  BEGINN
Local:
word xdial, ydial, wdial, hdial;
word ret_val;
Global:
word handle;

   v_hide_c(handle)

   form_dial(0, 0, 0, 0, 0, xdial, ydial, wdial, hdial)

   form_dial(1, 0, 0, 0, 0, xdial, ydial, wdial, hdial)

   objc_draw(dialog_addr, ROOT, MAX_DEPTH,
             xdial, ydial, wdial, hdial)

   v_show_c(handle, reset)

   form_do(dialog_addr, ed_start)

   form_dial(2, 0, 0, 0, 0, xdial, ydial, wdial, hdial)

   form_dial(3, 0, 0, 0, 0, xdial, ydial, wdial, hdial)

   return(ret_val)

   END
```

Was ist, wenn wir mehrere Eingabefelder in unserer Dialogbox haben?

Dies geschieht genau analog zu einem Textfeld, nur müssen wir mehr definieren (wie oben). Stehen mehrere Textfelder beispielsweise untereinander, so empfiehlt es sich, im Resource Construction Set die Option "Sort" zu wählen. Ansonsten kann es passieren, daß beim Editieren der Cursor, wenn er ins nächste Feld soll, wild hin- und herspringt.

Fehleralarm!

Im Betriebssystem des Atari gibt es in diesem Zusammenhang einen schweren Fehler. Angenommen, bei 'PVALID' lassen Sie nur Großbuchstaben zu oder nur Zahlen. Dann stürzt der Rechner **sofort** ab, wenn der Benutzer ein Underscore ("_") eintippt.

Es gibt zwei Möglichkeiten, dies Problem zu lösen:

1.) Sie lassen den Underscore als Zeichen zu - das geht, wenn Sie bei PVALID "X" oder "F" oder "P" angeben. Falls Sie nur Zahlen zulassen wollen, müssen Sie im Programm testen, ob die Eingabe auch nur daraus besteht.

2.) Eleganter ist die Methode, die Tastatur umzuprogrammieren. Dies geht mit der Xbios-Funktion *Keytbl()* (siehe unter der entsprechenden Beschreibung). Sie können anstatt des Underscores etwa ein Blank oder eine Null in die Tabelle eintragen (= Zeichen existiert nicht). Mit der Funktion "Bioskeys()" kann die ursprüngliche Tabelle wiederhergestellt werden. Dies muß vor Programmende geschehen, da nach Verlassen des Programms dieses aus dem Speicher gelöscht wird. Die Zeiger auf die Tastaturtabelle weisen dann auf lauter Nullen (und Sie können nichts mehr eingeben...)!

Directory

Das D. folgt immer hinter der »FAT und ist eigentlich sieben Sektoren lang (das ist abhängig von der Menge der erlaubten Einträge, der Defaultwert ist 112). Das Ende des Directory ist erreicht, wenn der Eintrag 32 Nullen enthält.
Wenn sie einen Eintrag löschen, wird er als gelöscht markiert und es steht dann am Anfang des Namens: $E5

Ein Directoryeintrag ist 32 Bytes Lang:

Byte	Bedeutung
$00 - $0A	enthält den Namen der Datei ohne Punkt
$0B	enthält das Datei-Attribut. Siehe dazu unter der »GEMDOS-Funktion *Fsfirst()* nach.
$0C - $15	reserviert
$16 - $17	enthält die Zeit im DOS-Format. Siehe dazu unter der »GEMDOS-Funktion *Tgettime()* nach
$18 - $19	enthält das Datum im DOS-Format. Siehe dazu unter der »GEMDOS-Funktion *Tgetdate()* nach
$1A - $1B	Start-Cluster im »Intel-Format
$1C - $1F	Länge der Datei im »Intel-Format Intel-Format: $11223344 68000'er-Format:$44332211

Diskettenaufbau

Der Atari unterstützt einseitige und doppelseitige Disketten und ist kompatibel zum MS-DOS Diskettenformat, allerdings mit einem Unterschied: der Atari hat 80 Track's und die meisten MS-DOS Rechner haben 40 Track's. Der »Bootsektor liegt immer auf Track Null und Sektor eins. Bei doppelseitigen Disketten wird zuerst der Track von Seite Null und dann der Track von Seite eins gelesen usw.. Um mehr über den Diskettenaufbau zu erfahren, sollten Sie unter den Stichwörtern: »Bootsektor, »Directory und »FAT nachschlagen.

Aufbau:

Sektor	Bedeutung (bei normalen Diskettenformat)
1	Bootsektor
2 - 6	FAT 1
6 - 11	FAT 2
12 - 18	Directory
13 - ...	Datensektoren

Disktransferaddress

Siehe zu dem Aufbau von D. unter der »Gemdos-Funktion *Fsfirst()* nach.

DMA Direct-Memory-Access

Mit direktem Speicherzugriff bezeichnet man ein Art des Speicherzugriffs, die nicht von der CPU durchgeführt wird, sondern von einem oder mehreren anderen Chips. Bei diesen Chips

handelt es sich entweder um sogenannte DMA-Controller, die zum Verschieben von Speicherbereichen dienen, während die CPU gerade anderweitige Aufgaben übernimmt. Ansonsten gibt es noch »I/O-Chips, die ihre Daten nicht an die CPU liefern bzw. von ihr erhalten, sondern die die Daten direkt in den Speicher schreiben und auch von dort holen. Dieses Vorgehen führt natürlich zu einer gewaltigen Entlastung der CPU, die sich jetzt sinnvolleren Aufgaben widmen kann.

DMA-Chip

Im Atari ST ist ein Spezialchip enthalten, der gewisse »DMA Funktionen übernimmt. Er besitzt fünf Register. Das erste Register ist unter der Adresse $FF8604 erreichbar. Es hat eine Länge von 16 Bit. Wenn Bit 4 des Statusregisters ($FF8606) gelöscht ist, befindet sich bei $FF8604 ein Register des »FDC 1772. Ist das Bit jedoch gesetzt, befindet sich bei $FF8604 das Sector-Count-Register. Die Zahl in diesem Register gibt an, wieviele Sektoren a 512 Byte transferiert werden sollen. Der kleinste Wert ist Null, der größte $00FF=255.

Das zweite Register ist ebenfalls 16 Bit lang und stellt das Status- und Steuerregister des DMA-Chips dar. Es hat folgenden Aufbau:

```
Bit 0 - Nicht benutzt.
Bit 1 - 0=Pin CA1 wird auf Low gesetzt.
        1=Pin wird High.
Bit 2 - 0=Pin CA2 wird auf Low gesetzt.
        1=Pin wird High.
Bit 3 - 0=Zugriff auf »FDC 1772.
        1=Zugriff auf Harddiskcontroller.
Bit 4 - 0=Auswahl des Status-Registers des FDC's.
        1=Auswahl des Sektor-Count-Registers.
Bit 5 - Nicht benutzt.
Bit 6 - 0=DMA einschalten.
        1=DMA ausschalten.
Bit 7 - Registerauswahl im FDC bzw. HDC.
Bit 8 - 0=FDC/HDC-Register lesen.
        1=FDC/HDC-Register schreiben.
```

Der DMA-Chip besitzt drei weitere Register mit einer Breite von jeweils acht Bit. Diese Register nehmen die Adresse auf, auf die im Speicher zugegriffen werden soll. Das Register bei $FF8609 nimmt die Bits 23 bis 16 der Adresse auf, das Register bei $FF860B die Bits 15 bis 8 und das Register bei $FF860D die Bits 7 bis 0. Bei der Programmierung dieser Register ist zu beachten, daß zuerst das Register $FF860D, dann das Register $FF860B und zuletzt das Register $FF8609 beschrieben werden muß.

Um einen DMA-Transfer durchzuführen, muß zuerst das Betriebssystem des Atari ST daran gehindert werden, während des »V-Blank Interrupts auf die Floppy zuzugreifen um festzustellen, ob die Diskette gewechselt wurde. Dies geschieht durch Schreiben von $FFFF in Adresse $00043E. Nach dem DMA-Transfer **muß** diese Speicherstelle dann wieder gelöscht werden. Im nächsten Schritt muß der Ringpuffer des DMA-Chips gelöscht werden. Dies geschieht durch Schreiben von $00BE gefolgt von $005A in das Steuerregister ($FF8606). Dann wird die gewünschte Adresse im Hauptspeicher in die drei genannten Register geschrieben: Bit 7-0 --> $FF860D, Bit 15-8 --> $FF860B und Bit 23-16 --> $FF8609.

Um ein Register des »FDC 1772 anzusprechen, sind zwei Zugriffe erforderlich. Der erste bestimmt das Register, der zweite schreibt in das oder liest aus dem Register. Die Auswahl erfolgt, indem einer der folgenden Werte in das Statusregister ($FF8606) geschrieben wird:

$0080 - Selektiert das Steuer- bzw. Statusregister des FDC.
$0082 - Selektiert das Track-Register.
$0084 - Selektiert das Sektor-Register.
$0086 - Selektiert das Datenregister.

Soll ein DMA-Transfer vom Speicher zur Floppy erfolgen, dann muß bei jedem der genannten Werte Bit 8 gesetzt werden.

Drop-Down-Menü

In »GEM besteht ein Menü aus einer Textzeile am oberen Bildschirmrand. Sobald der Mauscursor eines der Worte berührt, erscheint darunter ein weißer Kasten, der ein Untermenü enthält. Wenn der Mauszeiger einen der Unterpunkte streift, so wird dieser in invertierter Schrift dargestellt. Ist der entsprechende Menüeintrag in heller Schrift geschrieben, so geschieht dies nicht. Der Menüpunkt ist nicht wählbar. Der Menütext, der weiß auf schwarz geschrieben steht, kann durch Drücken der linken Maustaste gewählt werden (das Untermenü wird dann entfernt). Klickt man mit der Maus eine andere Stelle am Bildschirm (oder einen nicht wählbaren Menüpunkt) an, so verschwindet das Drop-Down-Menü, ohne das etwas ausgewählt wurde.

Da die Menüwahl in (fast) allen »GEM-Programmen gleich aussieht, handelt es sich hierbei um eine konsistente Benutzerschnittstelle. Dies ist von Vorteil für Programme, die nur eine kurze Einarbeitungszeit haben sollen.

Die Datenstruktur, die den Drop-Down-Menüs zugrunde liegt, ist (fast) dieselbe wie bei den »Objekten. Daher sollte das Stichwort »Objekte zuerst gelesen werden!

Zunächst ein paar Anmerkungen zu den im folgenden verwendeten Begriffen:
Die Textzeile am oberen Bildschirmrand bezeichne ich als Menüleiste, die darin stehenden Wörter sind Menütitel. Die unter einem Menütitel erscheinende Box erkläre ich zum Untermenü oder auch zur Menü-Box und der darin stehende Text ist ein Menüeintrag (oder auch Menüpunkt).

Im allgemeinen besteht ein Drop-Down-Menü nur aus Text (»Objekte vom Typ "G_STRING"). Diese haben exakt die Breite der Menü-Box. Wäre dies nicht der Fall, so würde der ganze Kasten invertiert, falls die Maus die "unbedeckte" Stelle berührt! Übrigens ist es möglich, andere »Objekte als vom Typ "G_STRING" in den Menüs zu verwenden. Darüber später mehr.

Bei der Definition der Drop-Down-Menüs ist zu beachten, daß es eine Reihe von Konventionen gibt. So wird der Menütitel ganz links meistens mit "Desk" bezeichnet. Der erste Menüeintrag von "Desk" ist der "Info-Eintrag" (gibt bei Wahl Informationen über das Programm preis). Insgesamt hat das Menü "Desk" exakt acht Einträge: den "Info-Eintrag", einen trennende Zeile aus nicht wählbaren Strichen sowie sechs leere Einträge für eventuell vorhandene »Desk-Accessories.

Die trennende Zeile ist übrigens deshalb nicht wählbar, weil bei dem entsprechenden »Objekt die Eigenschaft "DISABLED" vorhanden ist.

Die Konstruktion des Menüs sollte der Einfachheit halber mit einem "Resource Construction Set" vorgenommen werden. Dies verhindert auch Fehler beim Aufbau der Menüstruktur. Auch wenn diese aus einem »Objektbaum besteht, so gibt es doch eine Reihe von Einschränkungen. So kann z. B. die Größe einer Menü-Box ein Viertel des Bildschirms nicht überschreiten, da »GEM den Hintergrund für die Box in einem Puffer zwischenspeichert. Dieser faßt nur einen Viertel des Bildschirminhalts. Ist das Menü größer, so kann das System "abstürzen".

Die Benutzung der Drop-Down-Menüs ist einfacher als die der »Dialogboxen. Zunächst muß die Adresse des »Objektbaums ausfindig gemacht werden, die das Menü enthält (dies setzt voraus, daß das Resource-File wie unter »Dialogbox näher beschrieben geladen wurde!). Wie bei den »Dialogboxen erläutert, geschieht dies so:

```
    rsrc_gaddr(0, MENUNAME, &menu_address);
word            0, MENUNAME;
word *                      menu_address;
```

word MENUNAME Dies ist der symbolische Name des Menübaums, den Sie im "Resource Construction Set" vergeben haben.

word *menu_address Dies ist die Adresse der Variablen, die nachher die Adresse des Menübaums enthalten soll (also ein Zeiger auf ein Langwort).

Nun genügt ein Aufruf der Routine »*menu_bar()*:

```
    menu_bar(menu_adress, 1);
word *      menu_adress;
word                    1;
```

Die Eins bedeutet, daß »AES die Menüleiste anzeigen soll. Steht dort eine NULL, so wird die Menüleiste gelöscht. Dies sollte immer gemacht werden, bevor das Programm verlassen wird!

Nun kümmert sich »AES um die Interaktion des Benutzers mit dem Menü. »AES sorgt dafür, daß das Menü dargestellt wird, wenn der Benutzer einen Menütitel mit der Maus berührt (genauer gesagt ist dafür der Screen-Manager zuständig). Wird ein Menüpunkt ausgewählt, so sendet »AES eine »Nachricht an das Programm: "MN_SELECTED". Siehe auch »Message!

Um die »Nachricht zu empfangen, muß das Programm auf das entsprechende Ereignis warten. Siehe dazu auch »*evnt_mesag()* und vor allem »*evnt_multi()*. An diese beiden Routinen (meistens wird im Programm »*evnt_multi()* verwendet) muß ein Zeiger auf ein acht Worte unfassendes Array übergeben werden (bezeichnen wir das Array mit dem Namen "msg"). In msg(0) steht die Art der Nachricht (hier: "MN_SELECTED"), in msg(3) der angesprochene Menütitel und in msg(4) der gewählte Menüeintrag (wurde nichts gewählt, so bekommt das Programm auch keine »Nachricht).

Menütitel und Menüeinträge sind im "Resource Construction Set" mit symbolischen Namen versehen worden. Nun muß das Programm nur noch mit einem "switch-case" unterscheiden, welcher Menüeintrag angeklickt worden ist, um dann die entsprechenden Maßnahmen zu ergreifen (die symbolischen Namen sind die Indexnummern der »Objekte im Menübaum).

Nachdem der Benutzer einen Menüeintrag gewählt hat, bleibt der Menütitel invertiert. Dies läßt sich rückgängig machen:

```
    menu_tnormal(menu_adress, msg(3), 1);
word *           menu_adress;
word                         msg(3), 1;
```

In 'msg(3)' steht (wie oben erläutert) die Objektnummer des Menüeintrags. Wenn der letzte Parameter NULL ist, so wird der Menütitel invertiert. Dies kann dazu verwendet werden, um ein Menü hervorzuheben, wenn eine der Menüeinträge anders (z. B. über Tastatur) aktiviert wurde.

Manchmal ist es nützlich, einen Menüeintrag zu kennzeichnen. Dies kann mit »*menu_icheck()* erreicht werden. Der Menütext kann mittels »*menu_text()* geändert werden.

Ein Menüeintrag läßt sich aktivieren oder deaktivieren mit der Routine »*menu_ienable()*. Ein deaktivierter Menüpunkt wird in heller Schrift dargestellt und kann nicht gewählt werden. Dies ist nützlich, wenn man Fehler verhindern möchte, **bevor** sie entstehen.

Alle hier erwähnten Routinen sind Bestandteil der »Menu Library. Sehen Sie für weitere Informationen unter diesem Stichwort nach bzw. bei den oben beschriebenen Routinen.

DTA

Siehe zu dem Aufbau von D. unter der »Gemdos-Funktion *Fsfirst()* nach.

Elternobjekt
Siehe »Objekt

Environment String
Dieser String gibt eine bestimmte Arbeitsumgebung eines Programmes an. Der Zeiger auf diesen String steht in der »Base-Page des Programmes selbst. Dieser String ist entweder leer, d.h. er besteht nur aus einem Nullbyte, oder er enthält bestimmte Parameter, die vom Betriebssystem ausgewertet werden wenn das Programm eine entsprechende Routine aufruft. Der E. enthält z.B. »Pfadnamen, unter denen ausführbare Programme gesucht werden: "PATH= \USR\PROGS".

Ereignis
siehe »Event

Escape-Sequenz
Eine E. ist eine Folge von »ASCII-Zeichen, die mit einem Escape Zeichen ($1B=27) beginnt. Diese Sequenzen dienen der Steuerung von Geräten wie z.B. einem Terminal oder einem Drucker. Das *eingebaute* Terminal des Atari ST versteht die E. des VT52-Terminals (»Bildschirmsteuerung).

Event
(auf deutsch "Ereignis"): Ein Computerprogramm muß auf die Eingaben (Aktionen) **und** Unterlassungen des Benutzers reagieren. Jede der möglichen Benutzereingaben stellt ein Ereignis dar, aufgrund dessen ein bestimmter Programmteil ausgeführt werden soll. Ein solches Event kann z. B. sein:

a.) Das Verstreichen einer bestimmten Zeit (in der z. B. der Benutzer auf etwas hätte reagieren sollen)
b.) Das Drücken einer Maustaste
c.) Der Mauspfeil erreicht oder verläßt einen bestimmten Bereich
d.) Eine bestimmte Taste wurde gedrückt

Zur genaueren Beschreibung siehe auch die »Event Library.

Event Library
Um flexibel auf Benutzereingaben reagieren zu können, verfügt GEM über eine Reihe von Routinen, die auf »Events (= Ereignisse) warten. Nicht nur Eingaben werden als Ereignisse bezeichnet, sondern z. B. auch das Verstreichen einer bestimmten Zeit.

Auf folgende Ereignisse kann GEM reagieren:

a.) Eingabe über »Tastatur
b.) Drücken einer »Maustaste
c.) Bewegen des »Mauszeigers in ein bestimmtes Gebiet hinein oder aus einem Gebiet heraus
d.) Auswahl eines »Drop-Down-Menüs
e.) Anklicken eines Kontrollfeldes an einem Fensterrand
f.) Keine Eingabe des Benutzers, wenn diese erwartet wird (bestimmte Zeit ist verstrichen)

Wenn mehrere Applications gleichzeitig laufen, so ist es wesentlich effizienter, wenn diese via »GEM auf ein Ereignis warten, als das permanent zwischen den Programmen hin- und hergeschaltet wird und die Programme jedesmal selbst testen, ob etwas passiert ist. Interessant dabei ist vor allem, daß ein Programm auf eine beliebige Kombination von Ereignissen warten kann oder nur auf eines.

Im folgenden ein kurzer Überblick über die entsprechenden Routinen:

»*evnt_button()*: Warten auf das Drücken einer Maustaste. Beim Aufruf kann festgelegt werden, auf welche Maustaste reagiert werden soll - die linke, rechte oder beide Maustasten.

»*evnt_dclick()*: Hiermit kann die Geschwindigkeit eingestellt werden, die für ein Doppelklick mit der Maus notwendig ist, oder die eingestellte Geschwindigkeit kann abgefragt werden.

»*evnt_keybd()*: Warten auf einen Tastendruck - die Routine gibt den »ASCII-Code und den »Scan-Code der gedrückten Taste zurück, nicht jedoch den Zustand der Shift- Alternate- Control- und Capslock-Taste. Letzteres kann abgefragt werden mit: »*graf_mkstate()* (»GEM-Routine), *Kbshift()* (»BIOS-Routine). Die hier erwähnten Routinen warten jedoch im Gegensatz zu »*evnt_keybd()* nicht, sondern liefern gleich den entsprechenden Wert zurück!

»*evnt_mesag()*: Warten auf eine Nachricht von »AES (s. u.).

»*evnt_mouse()*: Warten darauf, ob der Mauszeiger ein bestimmtes Gebiet auf dem »Bildschirm erreicht oder verläßt.

»*evnt_multi()*: Warten auf irgendeine Kombination der obigen »Ereignisse.

»*evnt_timer()*: Warten auf das Verstreichen einer bestimmten Anzahl von Millisekunden.

Evnt_button

Zur »Event-Library gehörende Routine.

```
$15 = 25    ev_breturn = evnt_button (ev_bclicks, ev_bmask,
word        ev_breturn,               ev_bclicks, ev_bmask;
                                      ev_bstate, &ev_bmx,
word                                  ev_bstate;
                                      &ev_bmx, &ev_bmy,
word *                                ev_bmx, ev_bmy
                                      &ev_bbutton, &ev_bkstate)
word *                                ev_bbutton,  ev_bkstate;
```

Hiermit kann das Programm auf ein Drücken einer oder mehrerer »Maustasten warten. Theoretisch kann dies einer von 16 (!) Mausknöpfen sein, wobei die Abfrage auch maskiert werden kann. Die Routine kehrt zurück, sobald eine der Tasten gedrückt **oder losgelassen** wurde - je nachdem, was gewünscht wird. Die gesetzten Bits im Rückgabewert zeigen dabei an, welche der »Maustasten betätigt wurden: ein gesetztes Bit signalisiert eine gedrückte Taste. Das niederwertigste Bit (Bit 0) bezeichnet dabei die linke Taste, Bit 1 die rechte Taste usw. (sofern vorhanden). Zusätzlich ist es möglich, anzugeben, wie oft die Taste gedrückt worden sein soll. Die Geschwindigkeit, mit der die Klicks hintereinander erfolgen müssen, kann mit »*evnt_dclick()* gesetzt werden. Ebenso erhält man den Tastatur-Status zurück, d. h. ob »Shift-»Alternate-»Control- oder »Capslock gedrückt wurde.

Bedingt durch die Anzahl der Rückgabewerte müssen Zeiger auf Variable übergeben werden, damit die Routine die Werte bei der aufrufenden Funktion direkt verändern kann.

Eingabeparameter:

word ev_bclicks — Die Anzahl von Knopfdrücken, die erfolgen muß (innerhalb einer vorgegebenen Zeit), damit die Routine zurückkehrt. Dabei muß man anmerken, daß schon ein Doppelklick im allgemeinen Anfänger überfordert. Es lassen sich übrigens sogar dreifach - oder zehnfach-Klicks verlangen - was nur für das Training von Fingerakrobaten und das Geschäft von Mausherstellern nutzbringend ist...

word ev_bmask — Maske für Mausknöpfe: jedes gesetzte Bit signalisiert, daß auf das Drücken der zugehörigen Maustaste gewartet wird (Bit 0 = links, Bit 1 = rechts etc).

word ev_bstate — Gibt an, ob auf Drücken oder Loslassen reagiert werden soll. Bit gesetzt: auf Drücken warten, sonst auf Loslassen warten.

word *ev_bmx,
word *ev_bmy,
word *ev_bbutton,
word *ev_bkstate — Zeiger auf Variablen, deren Adressen für Rückgabewerte gebraucht werden (s. u.)

contrl(0) = 21
contrl(1) = 3
contrl(3) = 0

Ausgabeparameter:

word ev_breturn — Gibt an, wie oft die Maus gedrückt wurde (innerhalb der mit »*evnt_dclick()* eingestellten Zeit).

word ev_bmx — Die aktuelle X-Koordinate des Mauszeigers während des Knopfdrucks.

word ev_bmy — Y-Koordinate des Mauszeigers

word ev_bbutton — Status der Maustasten: Bit gesetzt = zugehörige Taste gedrückt (s. o.).

word ev_bkstate — Der Status der Tastatur bei der Rückkehr. Bit gesetzt bedeutet, daß die entsprechende Taste gedrückt wurde:

Bit	Bedeutung
0	Rechte Shifttaste
1	Linke Shifttaste
2	Controltaste
3	Alternatetaste
4	Caps-Lock-Taste

contrl(2) = 5
contrl(4) = 0

```
int_out(0)  =  ev_breturn
int_out(1)  =  ev_bmx
int_out(2)  =  ev_bmy
int_out(3)  =  ev_bbutton
int_out(4)  =  ev_bkstate
```

Evnt_dclick

Zur »Event-Library gehörende Routine.

```
$1A = 26     ev_dspeed = evnt_dclick(ev_dnew, ev_dgetset)
word         ev_dspeed,              ev_dnew, ev_dgetset;
```

Mit dieser Routine kann man herausfinden, wie schnell ein Doppelklick erfolgen muß, oder man kann diese Geschwindigkeit einstellen. In der Routine »*evnt_button()* gibt es einen Parameter ('ev_bclick', entspricht 'ev_mbclick' in »*evnt_multi()*), der angibt, wie oft die Maustaste gedrückt sein muß, damit die Routine zurückkehrt. Eben dies läßt sich mit dieser Routine setzen.

Eingabeparameter:

word ev_dnew	Dieser Parameter kann Werte von 0 bis 4 annehmen. Dies entspricht den Einstellungen "Langsam - 2 - 3 - 4 - Schnell" des Kontrollfeld-Accessories
word ev_dgetset	Gibt den Grund für diesen Routinenaufruf an: 1 = Neue Doppelklick-Geschwindigkeit setzen, 0 = augenblickliche Geschwindigkeit abfragen. Im letzteren Fall wird der Wert von 'ev_dnew' ignoriert.

```
contrl(0)   =  26
contrl(1)   =   2
contrl(3)   =   0

int_in(0)   =  ev_dnew
int_in(1)   =  ev_dgetset
```

Ausgabeparameter:

word ev_dspeed	Die Geschwindigkeit des Doppelklicks entweder der neugesetzte Wert (ev_dgetset = 1), oder der schon existierende ('ev_dgetset' = 0).

```
contrl(2)   =   1
contrl(4)   =   0

int_out(0)  =  ev_dspeed
```

Evnt_keybd

Zur »Event-Library gehörende Routine.

```
$14 = 20        ev_kreturn = evnt_keybd()
word            ev_kreturn;
```

Läßt das Programm auf eine Tastatureingabe des Benutzers warten. Die Routine kehrt zurück, wenn irgendeine Taste gedrückt worden ist.

Eingabeparameter: keine

```
contrl(0)   =  20
contrl(1)   =   0
contrl(3)   =   0
```

Ausgabeparameter:

word ev_kreturn — Enthält den Code der gedrückten Taste. Die unteren acht Bit enthalten der »ASCII-Code der gedrückten Taste, die oberen acht Bit bestehen aus dem »Scan-Code der Tastatur (kompatibel zu IBM). Dies ist anders als bei den »BIOS- oder »GEMDOS-Routinen zur Tastaturabfrage (dort wird ein Langwort zurückgegeben)! Der »ASCII-Code ist NULL, wenn eine »Funktionstaste, ‹Help›, ‹Undo› oder ähnliches gedrückt wurde.

```
contrl(2)   =   1
contrl(4)   =   0

int_out(0)  =  ev_kreturn
```

Evnt_mesag

Zur »Event-Library gehörende Routine.

```
$17 = 23    ev_mgresvd = evnt_mesag(&ev_mgpbuff)
word        ev_mgresvd
word *                              ev_mgpbuff;
```

Diese Routine kehrt zurück, wenn »AES eine Standard-16-Byte-Nach-

richt an die laufenden Application sendet. Über diesen Nachrichtenpuffer lassen sich Mitteilungen an andere Programme übermitteln. Es gibt eine Reihe von vordefinierten Nachrichten, die unter dem Stichwort »Event erklärt werden.

Eingabeparameter:

word *ev_mgp_buff Zeiger auf einen 16 Byte großen Nachrichtenpuffer. In diesem findet man die abgesandte Nachricht wieder.

```
contrl(0)   =  23
contrl(1)   =   0
contrl(3)   =   1

addr_in(0)  =  ev_mgpbuff
```

Ausgabeparameter:

word ev_mgresvd Immer gleich 1

```
contrl(2)   =   1
contrl(4)   =   0

int_out(0)  =  ev_mgresvd
```

Evnt_mouse

Zur »Event-Library gehörende Routine.

```
$16 = 22    ev_moresvd = evnt_mouse(ev_moflags, ev_mox, ev_moy,
word        ev_moresvd,              ev_moflags, ev_mox, ev_moy;
                                     ev_mowidht, ev_moheight,
word                                 ev_mowidth, ev_moheight;
                                    &ev_momx, &ev_momy,
word *                               ev_momx,  ev_momy:
                                    &ev_mobutton, &ev_mokstate)
word *                               ev_mobutton,  ev_mokstate;
```

Diese Routine wird aufgerufen, um festzustellen, ob der Benutzer mit dem Mauszeiger einen bestimmten Bereich des »Bildschirms erreicht oder verlassen hat. Dies wird (insbesondere in Zusammenhang mit einem »*evnt_multi()*-Aufruf) benötigt, um beispielsweise den »Mauszeiger auf Pfeil umzustellen, wenn er das aktive Fenster verläßt. U. U. reagieren die »Drop-Down-Menüs seltsam, wenn dies nicht der Fall ist.

Eingabeparameter:

word ev_moflags	Ein Flag, das anzeigt, ob die Routine beim Erreichen oder beim Verlassen des definierten Bereichs zurückkehren soll. 0 = Rückkehr beim Erreichen 1 = Rückkehr beim Verlassen
word ev_mox	X-Koordinate der linken oberen Ecke des Bereichs
word ev_moy	Y-Koordinate der linken oberen Ecke des Bereichs
word ev_mowidth	Breite des Bereichs in Pixeln
word ev_moheight	Höhe des Bereichs in Pixeln
word *ev_momx word *ev_momy word *ev_mobutton word *ev_mokstate	Zeiger auf Variablen, deren Adressen für Rückgabewerte gebraucht werden (s. u.)

```
contrl(0)   =  22
contrl(1)   =   5
contrl(3)   =   0

int_in(0)   =   ev_moflags
int_in(1)   =   ev_mox
int_in(2)   =   ev_moy
int_in(3)   =   ev_mowidth
int_in(4)   =   ev_ev_moheight
```

Ausgabeparameter:

word ev_moresvd	Dieser Wert ist immer 1
word ev_momx	Die X-Koordinate des Mauszeigers bei Erreichen oder Verlassen des angegebenen Bereichs
word ev_momy	Die Y-Koordinate des Mauszeigers bei Erreichen oder Verlassen des angegebenen Bereichs
word ev_mobutton	Der Status der Mausknöpfe bei Eintreten des Ereignisses (siehe »*evnt_button()*)

word ev_mokstate Der Status der Tastatur bei der Rückkehr. Bit gesetzt bedeutet, daß die entsprechende Taste gedrückt wurde:

Bit	Bedeutung
0	Rechte Shifttaste
1	Linke Shifttaste
2	Controltaste
3	Alternatetaste
4	Caps-Lock-Taste

```
contrl(2)   =   5
contrl(4)   =   0

int_out(0)  =  ev_moresvd
int_out(1)  =  ev_momx
int_out(2)  =  ev_momy
int_out(3)  =  ev_mobutton
int_out(4)  =  ev_mokstate
```

Evnt_multi

Zur »Event-Library gehörende Routine.

```
$19 = 25    ev_mwhich = evnt_multi(ev_mflags,
word                                ev_mflags,
                                    ev_mbclicks, ev_mbmask,
word                                ev_mbclicks, ev_mbmask,
                                    ev_mbstate,
word                                ev_mbstate,
                                    ev_mm1flags, ev_mm1x, ev_mm1y,
word                                ev_mm1flags, ev_mm1x, ev_mm1y,
                                    evmm1width, ev_mm1height,
word                                evmm1width, ev_mm1height,
                                    ev_mm2flags, ev_mm2x,evmm2y,
word                                ev_mm2flags, ev_mm2x,evmm2y,
                                    evmm2width,evmm2height,
word                                evmm2width,evmm2height,
                                    evmmgpbuff,
word *                              evmmgpbuff,
                                    ev_mtlocount, ev_mthicount,
word                                ev_mtlocount, ev_mthicount,
                                    &ev_mmox, &ev_mmoy,
word *                              &ev_mmox, &ev_mmoy,
                                    &ev_mmobutton, ev_mmokstate,
word *                              &ev_mmobutton, ev_mmokstate,
                                    &ev_mkreturn, &ev_mbreturn)
word *                              &ev_mkreturn, &ev_mbreturn
```

Diese nette, kleine Routine ist eine Zusammenfassung aller erklärten *evnt_...()*-Routinen. Während diese immer nur auf ein bestimmtes Ereignis warten, wartet diese Routine auf jede beliebige Kombination von Ereignissen zugleich. Sie werden diese Routine in (fast) jedem »GEM-Programm finden, welches Fenster oder Drop-Down-Menüs benutzt.

Die Parameter dieser Routine (es sind genau 23, die übergeben werden müssen) sind schon bei den anderen evnt-Routinen erläutert worden. Was hier z. B. 'ev_mmox' heißt, wird an anderer Stelle (»*evnt_mouse()*) 'ev_mox' genannt, also lediglich das zusätzliche 'm' hinter '_' ist anders. Erwähnenswert ist noch, daß anders als bei »*evnt_mouse()* zwei Bereiche definiert werden können, bei deren Erreichen und/oder Verlassen die Routine zurückkehrt.

Die Hauptschleife eines »GEM-Programms beginnt im allgemeinen mit einem *evnt_multi()*-Aufruf. Hinter diesem Aufruf folgen Programmstrukturen, die abtesten, welches der ausgesuchten Ereignisse eingetreten ist. Dort wird dann der für dieses Ereignis zuständige Programmteil aufgerufen.

Bei weiteren Fragen sehen Sie sich bitte die Erklärungen zu den anderen Routinen in diesem Kapitel, sowie die Erläuterungen zu: »Message und »Event an.

Eingabeparameter:

word ev_mmflags — Ein Flag, das angibt, auf welche (Kombination) der möglichen Ereignisse gewartet werden soll, indem das entsprechende Bit gesetzt wird (Bit 0 ist das am wenigsten bedeutsame Bit):

Bit	Ereignis	Flag
0	Tastatur-Ereignis	MU_KEYBD
1	Mausknopf-Ereignis	MU_BUTTON
2	Maus-Ereignis 1	MU_M1
3	Maus-Ereignis 2	MU_M2
4	Nachrichten-Ereignis	MU_MESAG
5	Zeit-Ereignis	MU_TIMER

Jede beliebige Kombination der Ereignisse ist möglich. Die Flags sind in "GEMDEFS.H" definiert.

word ev_mbclicks — Hiermit wird die Anzahl der Mausklicks angegeben, die zum Zurückkehren der Routine geschehen müssen. Die Geschwindigkeit der Klicks wird mit »*evnt_dclick()* oder dem Kontrollfeld eingestellt. Ob die Routine auf Drücken oder Loslassen reagieren soll, wird (für jeden Knopf getrennt) mit 'ev_mbstate' festgelegt. Auf welche Knöpfe überhaupt reagiert werden soll, das wird mit 'ev_mbmask' angegeben.

word ev_mbmask — Dieses Wort bildet eine Maske für das Maus-Ereignis. Jedes gesetzte Bit gibt an, auf welche Maustaste reagiert werden soll. Ist in 'ev_mbmask' Bit 0 gesetzt, so reagiert die Routine nur auf die linke Maustaste.

word ev_mbstate — Hier wird festgelegt, ob die Routine bei Drükken oder bei Loslassen zurückkehren soll (Bit = 1: entsprechender Mausknopf gedrückt, Bit = 0: Maus- knopf nicht gedrückt). Also: ein Mausknopf-Ereignis tritt ein, wenn folgendes wahr ist: ev_mbmask AND ev_mmobutton = ev_mbstate UND die Anzahl der Klicks dem mit 'ev_mbclicks' festgelegten Wert entspricht. Ansonsten: siehe »*evnt_button()*.

word ev_mm1flags — Dies ist der erste Parameter für ein Maus-Ereignis. Die Routine kehrt genau dann zurück, wenn 'ev_mm1flags' = 0 ist und der Mauszeiger den mit den folgenden vier Parametern definierten Bereich erreicht. Ist der Wert 1, so wird das Maus-Ereignis bei Verlassen des Bereichs generiert (siehe »*evnt_mouse()*).

word ev_mm1x — X-Koordinate von Bereich 1 (linke obere Ecke)

word ev_mm1y — Y-Koordinate von Bereich 1 (linke obere Ecke)

word ev_mm1width — Breite des Bereichs 1 in Pixeln

word ev_mm1height — Höhe des Bereichs 1 in Pixeln

word ev_mm2flags
word ev_mm2x
word ev_mm2y
word ev_mm2width
word ev_mm2height — Diese fünf Parameter haben exakt die gleiche Bedeutung der vorigen fünf - sie beziehen sich allerdings auf einen zweiten (vom ersten unab-

hängigen) Bereich. Somit ist es im Gegensatz zu »*evnt_mouse()* möglich, zwei verschiedene Bereiche zu definieren.

long mmgpbuff — Zeiger auf den 16-Byte großen Puffer, in den AES Nachrichten hineinschreibt (siehe »*evnt_mesag()* sowie die Definition der »Messages).

word ev_mtlocount — Das niederwertige Wort für ein Timer-Event.

word ev_mthicount — Das höherwertige Wort für ein Timer-Event Diese beiden Wörter zusammen ergeben das Langwort, das angibt, wieviele Millisekunden die Routine warten soll, bis ein Timer-Event ausgelöst wird.

word *+v_mmox
word *ev_mmoy
word *ev_mmobutton
word *ev_mmokstate
word *ev_mkreturn — Zeiger auf Variablen für Rückgabeparameter (siehe unten).

```
contrl(0)   =  25
contrl(1)   =  16
contrl(3)   =   1

int_in(0)   =  ev_mflags
int_in(1)   =  ev_mbclicks
int_in(2)   =  ev_mbmask
int_in(3)   =  ev_mbstate
int_in(4)   =  ev_mm1flags
int_in(5)   =  ev_mm1x
int_in(6)   =  ev_mm1y
int_in(7)   =  ev_mflags
int_in(8)   =  ev_mbclicks
int_in(9)   =  ev_mbmask
int_in(10)  =  ev_mbstate
int_in(11)  =  ev_mm1flags
int_in(12)  =  ev_mm1x
int_in(13)  =  ev_mm1y
int_in(14)  =  ev_mm1x
int_in(15)  =  ev_mm1y

addr_in(0)  =  ev_mmgpbuf
```

Ausgabeparameter:

word ev_mreturn — Gibt die Art des/der Ereignisse(s) an, das (die) zur Rückkehr der Routine geführt hat (haben) - (die entsprechenden Bits sind dann gesetzt):

Bit	Ereignis	Flag
0	Tastatur-Ereignis	MU_KEYBD
1	Mausknopf-Ereignis	MU_BUTTON
2	Maus-Ereignis 1	MU_M1
3	Maus-Ereignis 2	MU_M2
4	Nachrichten-Ereignis	MU_MESAG
5	Zeit-Ereignis	MU_TIMER

Dies entspricht der Bit-Belegung von 'ev_mmflags'!

word mmgpbuff[8] — In diesen Puffer schreibt AES seine Nachrichten hinein (siehe »*evnt_mesag()* sowie die Definition der Nachrichten).

word ev_mmox — Enthält die X-Koordinate der Maus, wenn ein Maus-Ereignis eingetreten ist

word ev_mmyx — Enthält die Y-Koordinate der Maus, wenn ein Maus-Ereignis eingetreten ist

word ev_mmobutton — Wenn ein Ereignis eintritt, steht in dieser Variablen der Zustand der Mausknöpfe (Bit gesetzt = entsprechender Mausknopf gedrückt, das niederwertigste Bit bezeichnet wie gehabt die Maustaste ganz links).

word ev_mmokstate — Der Status der Tastatur bei der Rückkehr. Bit gesetzt bedeutet, daß die entsprechende Taste gedrückt wurde:

Bit	Bedeutung
0	Rechte Shifttaste
1	Linke Shifttaste
2	Controltaste
3	Alternatetaste
4	Caps-Lock-Taste

word mkreturn — Der »ASCII-Code der gedrückten Taste in den unteren, der »Scancode der gedrückten Taste in den oberen acht Bit (anders als bei »GEMDOS oder »BIOS-Routinen!).

word ev_mbreturn Gibt an, wie oft die Maus gedrückt wurde (innerhalb der mit »*evnt_dclick()* eingestellten Zeit).

```
contrl(2)   =   7
contrl(4)   =   0

int_out(0)  =  ev_mreturn
int_out(1)  =  ev_mmox
int_out(2)  =  ev_mmoy
int_out(3)  =  ev_mmobutton
int_out(4)  =  ev_mmokstate
int_out(5)  =  evmkreturn
int_out(6)  =  ev_mbreturn
```

Evnt_timer

Zur »Event-Library gehörende Routine.

```
$18 = 24    ev_tresvd = evnt_timer(ev_tlocount, ev_thicount)
word        ev_tresvd,             ev_tlocount, ev_thicount;
```

Mit dieser Routine kann das Programm eine bestimmte Anzahl von Millisekunden warten. Es kommt in dieser Zeit auf die "»Not-Ready-List" und ist erst wieder an der Reihe, wenn die angegebene Zeit verstrichen ist oder ein anderes Ereignis eintritt.

Noch einen anderen Nutzen hat diese Routine: wenn eine »Application sehr zeitaufwendige Berechnungen ausführt, kann das dazu führen, daß die im Hintergrund laufenden Prozesse nicht mehr zum Zuge kommen. Mit anderen Worten: das Programm blockiert die anderen Programme. Um dies zu verhindern, sollte ab und zu ein Aufruf von *evnt_timer()* erfolgen, der Null Millisekunden wartet.

Eingabeparameter:

word ev_tlocount Niederwertiges Wort der Wartezeit

word ev_thicount Höherwertiges Wort der Wartezeit. Beide Worte zusammen ergeben ein Langwort, und dieses bestimmt den Zeitraum in Millisekunden.

```
contrl(0)   =  24
contrl(1)   =   2
contrl(3)   =   0

int_in(0)   =  ev_tlocount
int_in(1)   =  ev_thicount
```

Ausgabeparameter:

word ev_tresvd Immer gleich 1

contrl(2) = 1
contrl(4) = 0

int_out(0) = ev_tresvd

Exceptions

Engl.: Ausnahmen. Damit sind bestimmte Vorgänge im Computer gemeint, die nicht zum normalen Ablauf gehören. Bis auf den »Bus--Error werden alle E. von der 68000 CPU erkannt. Jeder dieser Ausnahmen ist ein bestimmter Vektor zugeordnet. In jeder der Adressen von $000004 bis $0003FF steht ein Vektor (Langwort), der auf eine bestimmte Routine zeigt, die für die Bearbeitung der Ursache dieses Aufrufes zuständig ist. Beim Atari ST zeigen alle Exceptionvektoren auf dieselbe Routine. Diese gibt die Nummer des Exceptionvektors in Form von Bömbchen aus, versucht, das fehlerhafte Programm normal zu beenden und kehrt dann zum Desktop zurück. Es gibt folgende Vektoren:

Vektor	Adresse	Zuordnung
0=$00	0=$0000	Dieses Langwort wird nach einem Reset in den Supervisor-Stackpointer geladen.
1=$01	4=$0004	Hier steht der Wert für den Program-Counter nach einem Reset.
2=$02	8=$0008	Bus-Error. Wird extern erzeugt.
3=$03	12=$000C	Address-Error. Zugriff auf ungerade Adresse
4=$04	16=$0010	Illegaler Opcode. Die CPU kennt diesen Befehl nicht.
5=$05	20=$0014	Division durch Null. Bei einem der Divisionsbefehle wurde durch Null geteilt.
6=$06	24=$0018	Der Vergleichswert beim CHK-Befehl lag außerhalb des erlaubten Bereiches.
7=$07	28=$001C	Die V-Flagge war beim TRAPV-Befehl gesetzt.
8=$08	32=$0020	Ein im »Usermode laufendes Programm hat versucht, einen nur im »Supervisormode erlaubten Befehl oder Speicherzugriff auszuführen.
9=$09	36=$0024	Das »Tracebit war gesetzt.
10=$0A	40=$0028	»Line-A Emulator. Die CPU ist auf einen Opcode getroffen, der mit %1010 beginnt.
11=$0B	44=$002C	»Line-F Emulator. Die CPU versucht, einen mit %1111 beginnenden Befehl auszuführen.
12=$0C	48=$0030	Reserviert.
13=$0D	52=$0034	Reserviert.
14=$0E	56=$0038	Format Fehler. Nur für CPU's 68010 und 68012.

Vektor	Adresse	Zuordnung
15=$0F	60=$0040	Reserviert bis 23=$17 Adresse 92=$005C.
24=$18	96=$0060	Spurious Interrupt. Zeigt einen »Bus-Error während einer Exceptionbearbeitung an.
25=$19	100=$0064	Level 1 »Interrupt Autovektor.
26=$1A	104=$0068	Level 2 »Interrupt Autovektor. Wird vom horital Blank benutzt.
27=$1B	108=$006C	Level 3 »Interrupt Autovektor.
28=$1C	112=$0070	Level 4 »Interrupt Autovektor. Wird vom »V-Blank benutzt.
29=$1D	116=$0074	Level 5 »Interrupt Autovektor.
30=$1E	120=$0078	Level 6 »Interrupt Autovektor.
31=$1F	124=$007C	Level 7 »Interrupt Autovektor (NMI).
32=$20	128=$0080	Trap #0 wird nicht benutzt.
33=$21	132=$0084	Trap #1 ruft das »GEMDOS auf.
34=$22	136=$0088	Trap #2 Ruft GEM auf (»AES und »VDI).
35=$23	140=$008C	Trap #3 wird nicht benutzt.
36=$24	144=$0090	Trap #4 wird nicht benutzt.
37=$25	148=$0094	Trap #5 wird nicht benutzt.
38=$26	152=$0098	Trap #6 wird nicht benutzt.
39=$27	156=$009C	Trap #7 wird nicht benutzt.
40=$28	160=$00A0	Trap #8 wird nicht benutzt.
41=$29	164=$00A4	Trap #9 wird nicht benutzt.
42=$2A	168=$00A8	Trap #10 wird nicht benutzt.
43=$2B	172=$00AC	Trap #11 wird nicht benutzt.
44=$2C	176=$00B0	Trap #12 wird nicht benutzt.
45=$2D	180=$00B4	Trap #13 Ruft Routinen des »BIOS auf.
46=$2E	184=$00B8	Trap #14 Ruft Routinen des »XBIOS auf.
47=$2F	188=$00BC	Trap #15 wird nicht benutzt.
48=$30	192=$00C0	Reserviert bis 63=$3F Adresse 252=$00FC.

Hier folgen die Non-Autovektoren. Diese werden folgendermaßen benutzt. Ein I/O-Chip löst einen Interrupt aus, der auf einem bestimmten Level (0-7) liegt. Im Atari ST ist dazu nur der »MFP 68901 in der Lage. Dieser legt einen Level 6 Interrupt an die CPU und einen Vektor auf die unteren 8 Bits des Datenbusses. Die 16 möglichen Interrupts, die der MFP auslösen kann, benutzen die ersten der zulässigen Non-Autovektoren. Es gilt folgende Zuordnung:

64=$40	256=$0100	(ausgeschaltet) Parallel port interrupt.
65=$41	260=$0104	(ausgeschaltet) V.24 Carrier Detect.
66=$42	264=$0108	(ausgeschaltet) V.24 CTS (Clear to send).
67=$43	268=$010C	(ausgeschaltet)
68=$44	272=$0110	(ausgeschaltet)
69=$45	276=$0114	200 Hz Systemtimer.
70=$46	280=$0118	Keyboard/MIDI (»ACIA 6850).
71=$47	284=$011C	(ausgeschaltet) Gepollter FDC/HDC.
72=$48	288=$0120	(ausgeschaltet) Horizontale Synchronisation.
73=$49	292=$0124	V.24 Sendefehler.

```
74=$4A  296=$0128  V.24 Sendepuffer ist leer.
75=$4B  300=$012C  V.24 Empfangsfehler.
76=$4C  304=$0130  V.24 Empfangspuffer ist leer.
77=$4D  308=$0134  (ausgeschaltet).
78=$4E  312=$0138  (ausgeschaltet) V.24 Ring indicator.
79=$4F  316=$013C  (ausgeschaltet) Monochrome Monitor detect.
```

Die restlichen 176 Non-Autovektoren werden im Atari ST nicht benutzt.

Extended Bios

Abk. für -Extended Basic Input Output System-. Häufig wird auch vom XBIOS geredet.

Das Extended Bios erleichtert das Ansprechen der Hardware des Atari ST. Alle Rückgabewerte des Extended Bios enthält das Register D0.

Ein Assembleraufruf sieht folgendermaßen aus:

```
move.w  #0,-(SP)        * Grafikauflösung
move.l  #$F8000,-(SP)   * Physische Bildschirmadresse
move.l  #$F8000,-(SP)   * Software Bildschirmadresse
move.w  #5,-(SP)        * Funktionsnummer von Setscreen()
trap    #14             * Aufruf des Extended Bios
add.l   #12,SP          * Stackpointer reparieren
```

Jeder Assemblerprogrammierer sollte wissen, daß der Trap-Handler nicht die Register D0-D2 und A0-A2 sichert.

Kurzübersicht der Funktionen des XBIOS mit Parametern:

Funktionsnummer	Funktion mit Parameter
$00 = 0	Initmous(type,param,vec)
$01 = 1	Ssbrk(amout)
$02 = 2	Physbase()
$03 = 3	Logbase()
$04 = 4	Getrez()
$05 = 5	Setscreen(LogLoc,physLoc,rez)
$06 = 6	Setpallete(palettePtr)
$07 = 7	Setcolor(colorNum,color)
$08 = 8	Floprd(buf,filler,devno,sectno,trackno,sideno, count)
$09 = 9	Florwr(buf,filler,devno,sectno,trackno,sideno, count)
$0A = 10	Flopfmt(buf,filler,devno,spt,trackno,sideno, interlv,magic,virgin)
$0B = 11	used-by-BIOS()

Funktionsnummer	Funktion mit Parameter
$0C = 12	Midiws(cnt,ptr)
$0D = 13	Mfpint(interno.vector)
$0E = 14	Iorec(devno)
$0F = 15	Rsconf(speed,flowctl,ucr,rsr,tsr,scr)
$10 = 16	Keytbl(unshift,shift,capslock)
$11 = 17	Random()
$12 = 18	Protobt(buf,serialno,disktype,execflag)
$13 = 19	Flopver(buf,filler,devno,sectno,trackno, sideno,count)
$14 = 20	Scrdmp()
$15 = 21	Cursconf(function,operand)
$16 = 22	Settime(datetime)
$17 = 23	Gettime()
$18 = 24	Bioskeys()
$19 = 25	Ikbdws(cnt,ptr)
$1A = 26	Jdisint(intno)
$1B = 27	Jenabint(intno)
$1C = 28	Giaccess(data,regno)
$1D = 29	Offgibit(bitno)
$1E = 30	Ongibit(bitno)
$1F = 31	Xbtimer(timer,control,date,vec)
$20 = 32	Dosound(ptr)
$21 = 33	Setptr(config)
$22 = 34	Kbdvbase()
$23 = 35	Kbrate(initial,repeat)
$24 = 36	Prtblk(zeiger)
$25 = 37	Vsync()
$26 = 38	Supexec(codeptr)
$27 = 39	Puntaes()

```
$00 = 0        Initmouse(type, param, vec)
word                     type;
long                           param, vec;
```

Mit dieser Funktion können Sie die »Maus manipulieren.

'type' gibt folgendes an:

type	Bedeutung
0	schaltet Maus aus
1	schaltet Maus ein und in den relativen Modus
2	schaltet Maus ein und in den absoluten Modus
3	unbenutzt
4	schaltet Maus ein und in den Tastaturmodus

'param' zeigt auf vier Bytes:

Offset	Bedeutung
0	wenn NULL ist die Y-Position unten und wenn eins ist die Y-Position oben
1	ist ein Kommando an den Tastaturprozessor zur Einstellung der Mausknöpfe (siehe dazu unter dem Begriff Tastaturprozessor)
2	Skalierungsfaktor in X-Richtung
3	Skalierungsfaktor in Y-Richtung

Wenn die Maus sich im absoluten Modus befindet, folgen noch vier Wörter:

Offset	Bedeutung
4	maximale X-Koordinate
6	maximale Y-Koordinate
8	absolute X-Mausposition
10	absolute Y-Mausposition

'vec' zeigt auf die Mausinterruptroutine.

Eingabeparameter:

word type	gibt den Modus der Maus an
long param	zeigt auf die Parameter zur Mauseinstellung
long vec	zeigt auf die Mausinterruptroutine

Ausgabeparameter: keine

$01 = 1 **zeiger = Ssbrk(amount)**
long zeiger, amount;

Mit dieser Funktion können Sie Speicher reservieren bevor das Betriebssystem installiert ist. Wenn die »TPA installiert ist, funktioniert es nicht mehr. Praktisch hat diese Funktion nur Sinn innerhalb von Bootroms.

'amount' gibt an wieviel Speicher von der oberen Grenze des Speichers angefordert werden soll.

'zeiger' ist die Startadresse des reservierten Speichers

Eingabeparameter:

long amount Menge des benötigten Speichers

Ausgabeparameter:

long zeiger Startadresse des Speichers

$02 = 2 **zeiger = Physbase()**
long zeiger;

Diese Funktion liefert in 'zeiger' die physikalische Adresse des Bildschirms zurück.

Eingabeparameter: keine

Ausgabeparameter:

long zeiger Startadresse des physikalischen Bildschirms

$03 = 3 **zeiger = Logbase()**
long zeiger;

Diese Funktion liefert in 'zeiger' die Softwareadresse des Bildschirms zurück. Diesen Bildschirm benutzt das Betriebssystem mit seinen Ausgabe- und Grafikfunktionen. Der Wert steht auch in der »Systemvariable **_v_bas_ad** ($44e).

Eingabeparameter: keine

Ausgabeparameter:

long zeiger Startadresse des Softwarebildschirms

```
$04 = 4          rez = Getrez()
word             rez;
```

Getrez() liefert die Auflösungsstufe zurück.

'rez' kann sein:

rez	Bedeutung
0	niedrigste Auflösung (320 * 200, 16 Farben)
1	mittlere Auflösung (640 * 200, 4 Farben)
2	hohe Auflösung (640 * 400, 2 Farben)

Eingabeparameter: keine

Ausgabeparameter:

```
word rez          Auflösungsstufe
```

```
$05 = 5          Setscreen(soft, phys, rez)
long                       soft, pyhs;
word                                  rez;
```

Setscreen() erlaubt das Verlegen des Bildschirms auf Adressen, die durch 256 teilbar sind, und das Einstellen der Grafikauflösung. Die Softwareadresse und die physische Adresse müssen nicht gleich sein. Es bietet sich an, dadurch einen Bildschirm aufzubauen und während dessen irgend einen anderen Bildschirm zu zeigen. Wenn Sie -1 als Parameter übergeben, wird der Parameter ignoriert.

'soft' zeigt auf den Bildschirm, den die Betriebssystemfunktionen benutzen.

'phys' zeigt auf den Bildschirm, den man auf dem Monitor sieht.

'rez' gibt an, welche Auflösungsstufe eingestellt werden soll.

Eingabeparameter:

```
long soft         Softwareadresse des Bildschirm oder -1
long phys         physikalische Bildschirmadresse oder -1
word rez          Grafikstufe oder -1
```

Ausgabeparameter: keine

$06 = 6 **Setpallete(palette)**
long palette;

Mit *Setpalette()* können Sie die Farbpalette einstellen, wobei Sie aus 512 Farben auswählen können. Mit dieser Funktion werden immer alle 16 Farbregister eingestellt.

'palette' zeigt auf 16 Wörter, in denen die Farben für die einzelnen Farbregister eingestellt werden.

Eingabeparameter:

long palette zeigt auf 16 Wörter

Ausgabeparameter: keine

$07 = 7 **oldcol = Setcolor(register, color)**
word oldcol, register, color;

Diese Funktion erlaubt das Verändern eines einzelnen Farbregisters.

'register' ist ein Wert zwischen 0 und 15 und gibt das Farbregister an, welches geändert wird.

'color' kann ein Wert zwischen Null und 511 sein und gibt damit die Farbe an. Wenn 'color' minus eins ist, wird das Farbregister nicht geändert.

'oldcol' ist die alte Farbe des Farbregisters.

Eingabeparameter:

word register Farbregister
word color Farbe oder minus eins

Ausgabeparameter:

word oldcol alter Inhalt des Farbregisters

```
$08 = 8         error = Floprd(buf, filler, devno, sectno,
word            error,                      devno, sectno,
long                          buf, filler;
                              trackno, sideno, count)
word                          trackno, sideno, count;
```

Mit *Floprd()* kann man Sektoren von Disketten einlesen.

'buf' zeigt auf einen Puffer, der 'count' mal 512 Byte groß sein muß, da ein Sektor 512 Byte Länge hat.

'filler' ist ein unbenutztes Langwort, dem irgendein Wert zugewiesen werden kann.

'devno' gibt an, welches Diskettenlaufwerk angesprochen werden soll:

devno	Bedeutung
0	Diskettenlaufwerk A:
1	Diskettenlaufwerk B:

'sectno' gibt an, bei welchem Sektor zu lesen begonnen werden soll.

'trackno' gibt an, von welchem Track die Sektoren eingelesen werden sollen.

'sideno' kann sein:

sideno	Bedeutung
0	obere Seite der Diskette
1	untere Seite der Diskette

'count' gibt an, wie viele Sektoren eingelesen werden sollen.

Es können folgende Fehler auftreten.

error	Bedeutung
0	Operation war erfolgreich
-2	Diskettenstation ist nicht betriebsbereit
-6	Track nicht gefunden
-8	Sektor nicht gefunden
-11	Lesefehler
-15	unbekanntes Gerät

Eingabeparameter:

long buf	Puffer für Sektoren
long filler	unbenutztes Langwort
word devno	Gerätenummer
word sectno	Sektornummer
word trackno	Tracknummer
word sideno	Seitennummer
word count	Menge der einzulesenden Sektoren

Ausgabeparameter:

word error	Fehler

```
$09 = 9         error = Flopwr(buf, filler, devno, sectno,
word            error,                      devno, sectno,
long                            buf, filler;
                                trackno, sideno, count)
word                            trackno, sideno, count;
```

Mit *Flopwr()* kann man Sektoren auf Disketten schreiben.

'buf' zeigt auf einen Puffer, der 'count' mal 512 Byte groß sein muß, da ein Sektor 512 Byte Länge hat.

'filler' ist ein unbenutztes Langwort, dem irgendein Wert zugewiesen werden kann.

'devno' gibt an welches Diskettenlaufwerk angesprochen werden soll:

devno	Bedeutung
0	Diskettenlaufwerk A:
1	Diskettenlaufwerk B:

'sectno' gibt an, bei welchem Sektor zu schrieben begonnen werden soll.

'trackno' gibt an, auf welchen Track die Sektoren geschrieben werden sollen.

'sideno' kann sein:

sideno	Bedeutung
0	obere Seite der Diskette
1	untere Seite der Diskette

'count' gibt an, wie viele Sektoren geschrieben werden sollen.

Es können folgende Fehler auftreten.

error	Bedeutung
0	Operation war erfolgreich
-2	Diskettenstation ist nicht betriebsbereit
-6	Track nicht gefunden
-8	Sektor nicht gefunden
-10	Schreibfehler
-13	Diskette schreibgeschützt
-15	unbekanntes Gerät

Eingabeparameter:

long buf	Puffer für Sektoren
long filler	unbenutztes Langwort
word devno	Gerätenummer
word sectno	Sektornummer
word trackno	Tracknummer
word sideno	Seitennummer
word count	Menge der zu schreibenden Sektoren

Ausgabeparameter:

word error	Fehler

```
$0A = 10        error = Flopfmt(buf, filler, devno, spt, trackno,
word            error,                       devno, spt, trackno,
long                          buf, filler,        magic;
                              sideno, interlv, magic, virgin)
word                          sideno, interlv,        virgin;
```

Diese Funktion erlaubt das formatieren einzelner Tracks.

'buf' zeigt auf einen Puffer, der bei neun Sektoren pro Track 8 KByte groß sein muß.

'filler' ist ein unbenutztes Langwort, dem irgendein Wert zugewiesen werden kann.

'devno' gibt an, welches Diskettenlaufwerk angesprochen werden soll:

devno	Bedeutung
0	Diskettenlaufwerk A:
1	Diskettenlaufwerk B:

'spt' gibt an wieviele Sektoren der Track haben soll. Maximal sind 10 möglich.

'trackno' gibt an, welcher Track formatiert werden soll.

'sideno' kann sein:

sideno	Bedeutung
0	obere Seite der Diskette
1	untere Seite der Diskette

'interlv' gibt an, welche Sektornummer der erste Sektor hat. Sie ist normalerweise eins.

'magic' muß die magische Nummer $87654321 enthalten.

'virgin' gibt an, womit die Sektoren gefüllt werden sollen. Der übliche Wert bei Atari ist $E5E5. Man sollte keine Bytes verwenden, die mit einem $F anfangen, da dies der Floppydiskkontroller beim Formatieren als Kommando interpretieren würde.

Beim Formatieren kann es sein, daß einige Sektoren nicht in Ordnung sind. Deshalb wird in einer Liste im Puffer zurückgegeben, welche Sektoren schadhaft sind. Die Liste wird mit einem Null-Wort beendet. 'buf' zeigt auf den Start dieses Puffers.

Es können folgende Fehler auftreten.

error	Bedeutung
0	Operation war erfolgreich
-2	Diskettenstation ist nicht betriebsbereit
-13	Diskette schreibgeschützt
-15	unbekanntes Gerät
-16	schadhafte Sektoren

Eingabeparameter:

long buf	Puffer für den Track
long filler	unbenutztes Langwort
word devno	Gerätenummer
word spt	Menge der Sektoren eines Tracks
word trackno	Tracknummer
word sideno	Seitennummer
word interlv	Erste Sektornummer
long magic	enthält die magische Zahl $87654321
word virgin	enthält den Wert, womit der Track aufgefüllt werden soll.

Ausgabeparameter:

word error	Fehler

$0B = 11 used-by-BIOS()

Diese Funktion wird vom »BIOS benutzt.

$0C = 12 Midiws(cnt, ptr)
word cnt;
long ptr;

Diese Funktion schickt eine Zeichenkette auf den »MIDI-Port hinaus.

'cnt' gibt die Menge der Zeichen minus eins an.

'ptr' zeigt auf die Zeichenkette.

Eingabeparameter:

word cnt	Menge der Zeichen minus eins
long ptr	Zeiger auf die Zeichenkette

Ausgabeparameter: keine

$0D = 13 **Mfpint(interno, vector)**
word interno;
long vector;

Mit *Mfpint()* können Sie die MFP-Interruptvektoren verändern.

'interno' gibt an welcher Vektor verändert werden soll (0 - 15).

'vector' zeigt auf die neue MFP-Interruptroutine.

Eingabeparameter:

word interno	Interruptnummer
long vector	Vektor der Interruptroutine

Ausgabeparameter: keine

$0E = 14 **zeiger = Iorec(devno)**
word devno;

Diese Funktion gibt einen Zeiger auf einen Record zurück, der den Puffer des seriellen Gerätes beschreibt.

'devno' kann sein:

devno	Bedeutung
0	RS 232
1	Tastaturprozessor
2	MIDI

Der Record ist folgendermaßen aufgebaut:

Typ	Name	Bedeutung
long	ibuf	Zeiger aud den Puffer
word	ibufsiz	größe des Puffers
word	ibufhd	head index
word	ibuftl	tail index
word	iduflow	low-water-mark
word	ibufhi	high-water-mark

Bei der RS 232 folgt der Record der Eingaberoutine der Record der Ausgaberoutine, der genauso aufgebaut ist.

Der Head-Index gibt an, wo das nächste Zeichen deponiert werden soll, und der Tail-Index gibt an, von wo das nächste Zeichen gelesen werden.

Wenn XON/XOFF oder RTS/CTS bei der RS 232 aktiviert sind und die Menge der Zeichen größergleich der high-water-mark sind, teilt der ST dem Sender mit, das er aufhören soll zu senden. Wenn dann wieder die low-water-mark erreicht ist, wird dem Sender mitgeteilt, daß er weiter senden kann.

Eingabeparameter:

word devno Gerätenummer

Ausgabeparameter:

long zeiger Zeiger auf den Record

$0F = 15 **Rsonf(speed, flowctl, ucr, rsr, tsr, scr)**
word speed, flowctl, ucr, rsr, tsr, scr;

Mit dieser Funktion können Sie die RS 232 konfigurieren. Wenn Sie als Parameter minus eins übergeben, wird der Parameter nicht verändert.

Mit 'speed' können Sie Baudrate der RS 232 einstellen:

speed	Baudrate
0	19200
1	9600
2	4800
3	3600
4	2400
5	2000
6	1800
7	1200
8	600
9	300
10	200
11	150
12	134
13	110
14	75
15	50

'flowctl' stellt die Flußkontrolle ein:

flowctl	Bedeutung
0	keine Flußkontrolle
1	XON/XOFF (^S/^Q)
2	RTS/CTS
3	XON/XOFF und RTS/CTS

Das Register 'ucr' ist auch wichtig für die RS 232 Einstellung:

Bit von ucr	Bedeutung
0	unbenutzt
1	1 = ungerade Parität, 0 = gerade Parität
2	1 = Parität eingeschaltet, 0 = Parität ausgeschaltet
3,4	stellt die Start- und Stop-Bits ein:

Bit 4	Bit 3	Start Bits	Stop Bits	Format
0	0	0	0	synchron
0	1	1	1	asynchron
1	0	1	1.5	asynchron
1	1	1	2	asynchron

5,6 Zeichenlänge:

Bit 5	Bit 6	Zeichenlänge in Bit
0	0	8
0	1	7
1	0	6
1	1	5

7 Geschwindigkeit 1 = 1/16 und 0 = volle Geschwindigkeit

'rsr','tsr' und 'scr' sind Register des MFP 68901

Eingabeparameter:

word speed	Baudrate
word flowctl	Flußkontrolle
word ucr	Register des MFP 68901
word rsr	Register des MFP 68901
word tsr	Register des MFP 68901
word scr	Register des MFP 68901

Ausgabeparameter: keine

```
$10 = 16        keytab = Keytbl(unshift, shift, capslock)
long            keytab,         unshift, shift, capslock;
```

Der Tastaturprozessor gibt nur Scancodes zurück. Deshalb braucht man Umrechnungstablellen zur Ermittlung der ASCII-Codes. Dabei hat man drei Möglichkeiten:

'unshift' zeigt auf eine 128 Byte lange Tabelle in der steht, welcher ASCII-Code dem Scancode zugeordnet wird, wenn keine Shift-Taste gedrückt wurde.

'shift' zeigt auf eine Tabelle, die 128 Byte lang ist. Hierbei werden den Scancodes die ASCII-Codes zugeordnet, die sich bei drücken der Shift-Tasten ergeben.

'capslock' zeigt auf eine Tabelle mit 128 Byte Länge. Diese Tabelle enthält die ASCII-Codes, die den Scancodes zugeordnet werden, wenn Capslock eingeschaltet ist.

Die ASCII-Codes sind in der Tabelle als Bytes abgelegt, wobei die Tabellen eine Länge von 128 haben müssen. Wenn Sie die Zeiger auf die oben beschrieben Tabellen nicht verändern wollen, müssen Sie minus eins übergeben.

'keytab' zeigt auf drei Langwörter, die Zeiger auf die 'unshift'-Tabelle, 'shift'-Tabelle und 'capslock'-Tabelle enthalten.

Die Veränderung können sie wieder rückgängig mit der »Extended Bios Routine *Bioskeys()* machen.

Eingabeparameter:

```
long unshift     Zeiger auf 'unshift'-Tabelle
long shift       Zeiger auf 'shift'-Tabelle
long capslock    Zeiger auf 'capslock'-Tabelle
```

Ausgabeparameter:

```
long keytab      Zeiger auf drei Langwörter
```

```
$11 = 17        zufall = Random()
long            zufall;
```

Man erhält von dieser Funktion eine 24 Bit lange Zufallszahl zurück.

Die Zufallszahl wird folgendermaßen errechnet:

S = S * C + K

wobei S die alte Zufallszahl ist, C = 3141592621 und K = 1 ist. Dann wird S acht Bits nach rechts geshiftet. S wird beim Start des Rechner der Inhalt der Systemvariablen *_frclock* ($466) zugewiesen, dadurch erhält man beim Start des Systems immer andere Zufallszahlen. Sie sollten nicht einzelne Bits der Zufallszahl testen. Sie sollten die Bits nicht einzeln testen, da die Bits Null bis 23 eine Wahrscheinlichkeit von 50% besitzen und die Bits 24 bis 31 immer Null sind.

Eingabeparameter: keine

Ausgabeparameter:

long zufall enthält die Zufallszahl

$12 = 18 Protobt(buf, serialno, disktype, execflag)

```
long                buf, serialno;
word                                 disktype, execflag;
```

Mit *Protobt()* kann man einen Bootsektor verändern bzw. erzeugen. Nach Aufruf dieser Funktion sollten Sie den Bootsektor wieder zurück auf Diskette schreiben. Der Bootsektor steht auf Spur Null Sektor eins.

'buf' zeigt auf einen Puffer, der 512 Byte groß ist und der entweder einen geladenen Bootsektor enthält oder irgendwelche Werte.

Mit 'serialno' können Sie die Seriennummer im Bootsektor einstellen:

serialno	Bedeutung
-1	Die Seriennummer soll nicht verändert werden
<$01000000	Es wird die Seriennummer in den Bootsektor gepackt, die 'serialno' enthält
>=$01000000	Es wird eine zufällige Seriennummer generiert

'disktype' erlaubt verschieden Diskettenformate einzustellen:

disktype	Bedeutung
-1	das Diskettenformat wird nicht verändert
0	40 Tracks, einseitig (180 K)
1	40 Tracks, doppelseitig (360 K)
2	80 Tracks, einseitig (360 K)
3	80 Tracks, doppelseitig (720 K)

'execfalg' ermöglicht einzustellen, ob der Bootsektor ausführbar ist, d.h., daß es möglich ist, ein Betriebssystem von der Diskette zu laden. 'execflag' kann folgende Werte enthalten:

execflag	Bedeutung
-1	'execflag' im Bootsektor soll nicht verändert werden 0
	Bootsektor ist nicht ausführbar 1
	Bootsektor ist ausführbar

Eingabeparameter:

long buf	Zeiger auf einen 512 Byte langen Puffer
long serialno	enthält Seriennummer
word disktype	enthält Informationen über den Diskettentyp
word execflag	sagt, ob Bootsektor ausführbar oder nicht

Ausgabeparameter: keine

```
$13 = 19        error = Flopver(buf, filler, devno, sectno,
word            error,                       devno, sectno,
long                          buf, filler;
                              trackno, sideno, count)
word                          trackno, sideno, count;
```

Mit *Flopver()* können Sie feststellen, ob einzelne Sektoren lesbar sind.

'buf' zeigt auf einen Puffer, der 1024 Byte groß ist und auf einer geraden Adresse beginnen muß.

'filler' ist ein unbenutztes Langwort, dem irgendein Wert zugewiesen werden kann.

'devno' gibt an, welches Diskettenlaufwerk angesprochen werden soll:

devno	Bedeutung
0	Diskettenlaufwerk A:
1	Diskettenlaufwerk B:

'sectno' gibt an, bei welchen Sektor mit dem Verify begonnen werden soll.

'trackno' gibt an, von welchem Track die Sektoren verifiziert werden sollen

'sideno' kann sein:

sideno	Bedeutung
0	obere Seite der Diskette
1	untere Seite der Diskette

'count' gibt an, wie viele Sektoren verifiziert werden sollen.

Es können folgende Fehler auftreten.

error	Bedeutung
0	Operation war erfolgreich
-2	Diskettenstation ist nicht betriebsbereit
-6	Track nicht gefunden
-8	Sektor nicht gefunden
-11	Lesefehler
-15	unbekanntes Gerät

Nach Ausführung dieser Funktion zeigt 'buf' auf eine Tabelle von Worten, die mit einer Null abgeschlossen wird. In ihr sind die Sektoren aufgeführt, die nicht einlesbar sind.

Eingabeparameter:

long buf	Puffer
long filler	unbenutztes Langwort
word devno	Gerätenummer
word sectno	Sektornummer
word trackno	Tracknummer
word sideno	Seitennummer
word count	Menge der zuverifizierenden Sektoren

Ausgabeparameter:

word error	Fehler

$14 = 20 Scrdmp()

Mit dieser Funktion können Sie den Bildschirm ausdrucken. Falls kein Drucker angeschlossen oder der Drucker OFF-Line ist, wird nach 30 Sekunden die Operation abgebrochen. Die Adresse der Routine, die bei dieser Funktion angesprochen wird enthält die »Systemvariable *dump_vec* ($502). Falls Sie eine Neue Hardcopyroutine installieren wollen, sollten Sie nicht die Routine in den »VBL hängen, sondern den Vektor auf der Adresse ($502) verändern, so daß er auf Ihre Routine zeigt, oder falls das nicht möglich ist sollten sie eine Null in die »Systemvariable *_prnt_cnt* ($4ee) laden, so wird dem Betriebssystem signalisiert, das die Alternate-Help-Taste gedrückt ist und dieses ist auch kompatibel zu den Hardcopyroutinen, die in den »VBL-Slot's eingebunden sind.

Eingabeparameter: keine

Ausgabeparameter: keine

$15 = 21 rate = Cursconf(function, operand)
word rate, function, operand;

Cursconf() erlaubt das konfigurieren, des Cursors.

'function' kann folgende Werte haben:

function	Bedeutung
0	Cursor ausschalten
1	Cursor einschalten
2	Cursor soll blinken
3	Cursor soll nicht blinken
4	Soll dem Cursorblinktimer den Wert von 'operand' zuweisen
5	gibt die Cursorblinkrate zurück

'operand' muß nur defeniert werden, wenn 'function' = 4 ist. Die Blinkrate errechnet sich folgendermaßen: operand/hertz
'hertz' kann folgende Werte annehmen:

hertz	Bedeutung
50	Bei Modulatoren oder Farbmonitoren
60	Bei Farbmonitoren
70	Bei monochromen Bildschirm

'rate' wird nur zurückgegeben wenn, 'function' = 5 ist. 'rate' ist die Cursorblinkrate.

Eingabeparameter:

word function Cursorstatus einstellen
word operand Cursorblinkrate

Ausgabeparameter:

word rate Cursorblinkrate

$16 = 22 Settime(datetime)
long datetime;

Diese Funktion ermöglicht die Uhrzeit und das Datum des Tastaturprozessors einzustellen. Die Zeit ist in den unteren 16 Bit und das Datum in den oberen 16 Bit. Das Format ist genauso wie bei den »Gemdos Funktionen *Tsettime()* und *Tsetdate()*.

Eingabeparameter:

long datetime Uhrzeit und Datum

Ausgabeparameter: keine

$17 = 23 datetime = Gettime()
long datetime;

Mit *Gettime()* können Sie die Uhrzeit und das Datum des Tastaturprozessors abfragen. Die Zeit ist in den unteren 16 Bit und das Datum in den oberen 16 Bit. Das Format ist genauso wie bei den »Gemdos Funktionen *Tgettime()* und *Tgetdate()*.

Eingabeparameter: keine

Ausgabeparameter:

long datetime enthält Uhrzeit und Datum

$18 = 24 Bioskeys()

Wenn Sie die Tastaturtabelle mit *Keytbl()* verändert haben, können Sie die Standardbelegung mit *Bioskeys()* wieder einstellen.

Eingabeparameter: keine

Ausgabeparameter: keine

$19 = 25 Ikbdws(cnt, ptr)
word cnt;
long ptr;

Mit *Ikbdws()* können Sie eine Zeichenkette an den Tastaturprozessor schicken.

'cnt' gibt an, wieviele Zeichen minus eins Sie an den Tastaturprozessor schicken wollen.

'ptr' zeigt auf die Zeichenkette, die Sie an den Tastaturprozessor schicken wollen.

Eingabeparameter:

word cnt	Menge der Zeichen minus eins
long ptr	zeigt auf die Zeichen

Ausgabeparameter: keine

$1A = 26 Jdisint(intno)
word intno;

Diese Funktion ermöglicht es einen Interrupt des MFP 68901 zu sperren.

'intno' gibt die Interruptnummer (0-15) an.

Eingabeparameter:

word intno	Interruptnummer

Ausgabeparameter: keine

$1B = 27 **Jenabint(intno)**
word intno;

Jenabint() ermöglicht es einen Interrupt des MFP 68901 freizugeben.

'intno' gibt die Interruptnummer (0-15) an.

Eingabeparameter:

word intno Interruptnummer

Ausgabeparameter: keine

$1C = 28 **ret = Giaccess(data, regno)**
byte ret;
word data, regno;

Mit dieser Funktion können Sie einzelne Register des Soundchips ansprechen.

In 'data' steht der Wert, der in ein Soundchipregister geschrieben werden soll.

'regno' gibt an, welches Register Sie ansprechen wollen. Wenn Sie einen Wert schreiben wollen müssen Sie das Register mit $80 OR-verknüpfen.

'ret' wird zurückgegeben, wenn ein Register gelesen werden sollte.

Eingabeparameter:

word data Datenbyte
word regno Registernummer

Ausgabeparameter:

byte ret Daten

$1D = 29 Offgibit(bitno)
word bitno;

Sie können hiermit ein Bit des Port A vom Soundchip löschen. Es wurde bisher immer falsch beschrieben, daß Sie die Bitnummer übergeben können. Es muß folgendermaßen aussehen:

Wenn Sie z.B.: das Bit 6 löschen wollen, müssen alle Bits außer dem Bit 6 gesetzt sein, weil der Wert mit dem Inhalt des Port A UND-veknüpft wird. Aufruf: Offgibit(191)

Bedeutung der Bits des Ports A:

Bit	Bedeutung
0	0 = Seite 1, 1 = Seite 0 der Floppy
1	0 = Selektieren , 1 = deselektieren des Laufwerks A
2	0 = Selektieren , 1 = deselektieren des Laufwerks B
3	RS 232 RTS
4	RS 232 DTR
5	Centronics Strobe
6	General Purpose Output
7	unbenutzt

Eingabeparameter:

word bitno enthält das Bitmuster, welches mit Port A Und-Verknüpft werden soll.

Ausgabeparameter: keine

$1E = 30 Ongibit(bitno)
word bitno;

Sie können hiermit ein Bit des Port A vom Soundchip setzen. Es wurde bisher immer falsch beschrieben, daß Sie die Bitnummer übergeben können. Es muß folgendermaßen aussehen:

Wenn Sie z.B.: das Bit 7 setzen wollen muß der Funktion 128 übergeben werden, weil der Wert mit dem Inhalt des Port A OR-verknüpft wird.

Die Bedeutung der Bits vom Port A sind bei *Offgibit()* beschrieben.

Eingabeparameter:

word bitno enthält den Wert, welcher mit Port A OR-verknüpft werden soll.

Ausgabeparameter: keine

$1F = 31 Xbtimer(timer, control, data, vec)
word timer, control, data;
long vec;

Hiermit können Sie die vier Timer des MFP 68901 ansprechen.

'timer' sind die Timernummern:

timer	Bedeutung
0	Timer A
1	Timer B
2	Timer C
3	Timer D

'control' enthält den Wert, der in das Timerkontrolregister geschrieben werden soll.

'data' ist der Wert, der in das Timerdatenregister geschrieben werden soll.

'vec' zeigt auf den Interrupthandler des Timers.

Folgendermaßen sind die Timer belegt:

Timer	Belegung
A	frei
B	Horizentrale Synchronisation
C	200 Hz Systemtimer
D	RS 232 Baudraten (der Interruptvektor ist frei)

Eingabeparameter:

word timer Timernummer
word control Wert für Kontrolregister
word data Wert für Datenregister
long vec Zeiger auf den Interrupthandler

Ausgabeparameter: keine

$20 = 32 Dosound(ptr)
long ptr;

Hiermit können Sie per Interrupt Musik abspielen lassen.

'ptr' zeigt auf eine Liste von Kommandos und Daten.

Kommandos Nummer 0 bis 15 besagen, daß das nächste Byte in das vom Kommando angegebene Soundchipregister geschoben werden soll.

Dem Kommando $80 folgt ein Byte, das in ein temporäres Register geschoben werden soll.

Dem Kommando $81 folgen drei Bytes. Das erste Byte gibt an, in welches Register das temporäre Register geladen werden soll. Das zweite Byte ist ein Zweierkomplement, welches zu dem temporären Register addiert wird. Dies wird solange gemacht, bis das dritte Byte mit dem temporären Register übereinstimmt.

Den Kommandos $82 - $FF folgt in einem Byte, wieviele Systemtimeraufrufe bei 50 Hz gewartet werden soll. Falls dieses Byte Null sein sollte, wird *Dosound()* beendet.

Eingabeparameter:

long ptr zeigt auf Kommandos und Daten

Ausgabeparameter: keine

$21 = 33 oldconf = Setprt(config)
word oldconf, config;

Mit *Setprt()* können Sie das Druckerkonfigurationsbyte verändern.

Wenn 'config' minus eins ist, wird das Konfigurationsbyte nicht verändert. Sonst haben haben die einzelnen Bits von 'config' folgende Bedeutung:

Bit	Bedeutung wenn Null	Bedeutung wenn 1
0	Matrixdrucker	Typenraddrucker
1	schwarz/weiß Drucker	Farbdrucker
2	Atari Drucker	*Epson* Drucker
3	normaler Modus	Schönschriftmodus
4	Centronics Port	RS 232 Port
5	Endlospapier	Einzelblatt
6	reserviert	
7	reserviert	
8	reserviert	
9	reserviert	
10	reserviert	
11	reserviert	
12	reserviert	
13	reserviert	
14	reserviert	
15	muß Null sein	

'oldconf' enthält das Konfigurationbyte, das vor dem Aufruf aktuell war.

Eingabeparameter:

word config Druckerkonfigurationsbyte

Ausgabeparameter:

word oldconf altes Druckerkonfigurationsbyte

$22 = 34 **zeiger = Kbdvbase()**
long zeiger;

Diese Funktion gibt einen Zeiger auf eine Tabelle mit Vektoren zurück.

'zeiger' zeigt auf folgende Tabelle mit Vektoren:

Offset	Name	Bedeutung
0	midivec	Zeigt auf eine Pufferroutine vom Bios. D0 enthält ein Zeichen, das von dem MIDI-Port gelesen wurde.
4	vkbderr	Dieser Vektor wird angesprungen, wenn ein Überlauf bei der Tastatur vorliegt
8	vmiderr	Dieser Vektor wird angesprungen, wenn ein Überlauf beim MIDI-Port vorliegt

Offset	Name	Bedeutung
12	statvec	Gibt den Status der Tastatur an
16	mousevec	Ist der Mousehandler, wird von GEM benutzt
20	clockvec	Vektor der Echtzeituhr
24	joyvec	Ist der Handler für den Joystick
28	midisys	Wird aufgerufen, wenn ein Zeichen von der MIDI-Schnittstelle erhältlich ist
32	ikbdsys	Wird aufgerufen, wenn ein Zeichen vom Tastaturprozessor erhältlich ist

Die Vektoren 'statvec', 'mousevec', 'clockvec' und 'joyvec' dürfen nur 1 Millisekunde Ausführungszeit beanspruchen und müssen mit dem Maschinenbefehl RTS abgeschlossen werden. Das Adressregister A0 enthält einen Zeiger auf das Packet, das von dem Tastaturprozessor zurückgegeben wurde.

Eingabeparameter: keine

Ausgabeparameter:

long zeiger Zeiger auf neun Vektoren

$23 = 35 rate = Kbrate(initial, repeat)
word rate, initial, repeat;

Mit dieser Funktion können Sie die Wiederholrate der Tastatur einstellen. Wenn ein Parameter -1 ist, wird es nicht verändert. Alle Werte basieren auf dem 50 Hz Systemtakt.

'initial' gibt an, wann die Wiederholfunktion der Tastatur ausgelöst werden soll.

'repeat' gibt die Wiederholgeschwindigkeit der Tastatur an.

'rate' enthält in Bit 0-7 die Wiederholgeschwindigkeit und in Bit 8-15 die Auslösegeschwindigkeit der Tastatur.

Eingabeparameter:

word initial Auslösegeschwindigkeit der Wiederholfunktion
word repeat Geschwindigkeit der Wiederholfunktion

Ausgabeparameter:

word rate Wiederhol- und Auslösegeschwindigkeit

$24 = 36 Prtblk(zeiger)
long zeiger;

Mit dieser Funktion können Sie den Bildschirminhalt drucken lassen. Sie ähnelt damit der Funktion *Scrdmp()* (*Scrdmp()* ruft auch diese Funktion auf).

'zeiger' zeigt auf eine Tabelle:

Offset	Bedeutung
0	Adresse des Bildschirms
4	Offset von der Bildschirmadresse
6	Breite des Bildschirms
8	Höhe des Bildschirm
10	Punkte am linken Rand auslassen
12	Punkte am rechten Rand auslassen
14	Auflösung des Bildschirms 0, 1 oder 2
16	Druckerauflösung 0 = Test
	1 = Qualität
18	Zeiger auf die Farbpalette
20	Druckertyp 0 = Atari S/W Matrixdrucker
	1 = Atari Farb Matrixdrucker
	2 = Atari S/W Typenraddrucker
	3 = Epson S/W Matrixdrucker
22	Druckerschnittstelle 0 = Centronics
	1 = RS 232
26-30	Zeiger auf Bitmaske 0 = Default vom Betriebssystem

Eingabeparameter:

long zeiger Zeiger auf Parametertabelle

Ausgabeparameter: keine

$25 = 37 Vsync()

Diese Funktion erlaubt ihnen, auf den nächsten Vertical-Blank Interrupt zuwarten. Sie ist sinnvoll zur Synchronisation von

Grafik mit dem Bildschirmaufbau.

Eingabeparameter: keine

Ausgabeparameter: keine

$26 = 38 Supexec(codeptr)
long codeptr;

Mit dieser Funktion können Sie Routinen im Supervisormode ausführen lassen. Sie ist sinvoll für Routinen, die direkt die Hardware ansprechen. Das Ende der Routine muß der Maschinenbefehl RTS sein. Von dieser Routine darf kein GEMDOS, BIOS und EXTENDED BIOS aufgerufen werden.

'codeptr' zeigt auf die Routine.

Eingabeparameter:

long codeptr Zeiger auf die Routine.

Ausgabeparameter: keine

$27 = 39 Puntaes()

Dies Funktion erlaubt das Abschalten des »AES. Sie funktioniert nur mit TOS-Versionen, die von Diskette geladen wurden, und führt danach einen Neustart des Systems aus. Falls das AES schon abgeschaltet ist, kehrt die Funktion wieder zurück.

Eingabeparameter: keine

Ausgabeparameter: keine

Extensions

Engl.: Erweiterung. Mit E. werden beim Atari ST meist die drei Zeichen gemeint, die bei einem Filenamen dem Punkt folgen. Folgende E. sind bedeutungsvoll:

ACC - Accessoryprogramm. Bei einem Reset sucht die System-initialisierung in der »Root-Directory nach Files mit dieser Endung. Die sechs Files, die zuerst gefunden werden, werden geladen und stehen dann als jederzeit benutzbares »Accessory zur Verfügung.
APP - Application. Ein unter »GEM laufendes Programm.
ASM - Kennzeichnet einen Assembler-Quelltext (»ASCII-File).
BAS - Kennzeichnet einen BASIC-Quelltext.
C - Kennzeichnet einen C-Quelltext.
CBL - Kennzeichnet einen Cobol-Quelltext.
COB - Kennzeichnet einen Cobol-Quelltext.
DIR - Hier verbirgt sich ein Subdirectory (wird oft weggelassen).
DOC - (Document) enthält ein Textfile, das im Gegensatz zu reinen ASCII-Files auch Steuerzeichen eines speziellen Textprogrammes enthalten kann. Ein ".DOC"-File enthält meist eine Anleitung oder eine Beschreibung (im Gegensatz zu einem ".TXT"-File).
FTN - Kennzeichnet einen Fortran-Quelltext (auch "FOR").
LEX - Files für dieses Lexikon...
MOD - Kennzeichent einen Modula-II Quelltext.
PRG - Zeigt ein unter GEM laufendes Programm an. Bei text-orientierten »Benutzeroberflächen ist diese Endung ebenfalls sinnvoll, da einige »Kommandointerpreter nur nach Files mit dieser Endung suchen.
TOS - Programme mit dieser Endung haben eine textorientierte Benutzeroberfläche. Beim Laden wird die Maus automatisch abgeschaltet und der Bildschirm wird gelöscht.
TTP - (TOS takes parameters) wie ein TOS-Programm, nur daß auch noch Parameter übergeben werden können.
TXT - Textfile; enthält zusätzlich zu den üblichen ASCII-Zeichen evtl. weitere Steuerzeichen, die meist nur von einem Textprogramm korrekt interpretiert werden können.

Farbregister

Der Atari ST kann in der niedrigsten Auflösungsstufe A0 bis zu 16 Farben gleichzeitig darstellen. Diese 16 Farben können aus insgesamt 512 verschiedenen Farbwerten ausgewählt werden. Jede der Farben wird in einem F. festgelegt. Diese F. werden ab $FF8240 adressiert, sind je ein Wort lang und haben folgenden Aufbau:

```
FEDBCA9876543210
-----rrr-ggg-bbb
     rrr   - Anteil der roten Farbe
         ggg   - Anteil der grünen Farbe
             bbb   - Anteil der blauen Farbe
```

Zum Belegen und Ändern dieser Farbregister stehen die beiden »XBIOS-Routinen $06 *Setpalette(paletteptr)* und $07 *Setcolor (colnbr, color)* zur Verfügung.

FAT

Abk. für File Allocation Table. In der FAT sind Informationen darüber enthalten, wo die einzelnen Dateien liegen.
Die FAT ist normalerweise fünf Sektoren lang und liegt hinter dem Bootsektor. Es existieren zwei FAT's.

Wenn Sie eine Datei löschen, werden die Cluster, die durch die Datei belegt waren, mit Nullen aufgefüllt, d.h die Cluster sind frei.

Die FAT beginnt mit folgender Bytekombination: $7F, $FF, $FF Danach folgen die Cluster. Jeder Eintrag in die FAT ist 12 Bit lang. Null markiert, daß der Cluster leer ist. In den 12 Bit steht immer der nächste Cluster der Datei. Falls der letzte Cluster der Datei erreicht ist, sind alle zwölf Bit gesetzt. Um genauere Information über den Aufbau der FAT zuerhalten, bitten wir Sie, im Anhang E die Abb. 1 zu betrachten.

FDC

FDC ist die Abkürzung für Floppydiskcontroller. Der FDC beim Atari ST ist der WD 1772, ein 5 1/4 Zoll FDC. Der FDC hat anscheinend einen Designfehler, denn der Befehl Read Track funktioniert nicht richtig. Beim Einlesen eines Tracks stimmen eigentlich nur die Header der einzelnen Sektoren, aber die Füllbytes und der Inhalt der einzelnen Sektoren stimmen meistens nicht.

Der WD 1772 kennt vier verschiedene Sektorgrößen: 128, 256, 512 und 1024 Byte lange Sektoren.

Alle Tracks mit 256, 512 und 1024 Byte Sektoren haben den gleichen Aufbau, und der Track mit 128 Byte Sektoren ist etwas anders aufgebaut (siehe dazu auch die Abb. 2 und Abb. 3 im Anhang E)

Folgende Kommandos kennt der WD 1772:

		Bits							
Typ	Kommando	7	6	5	4	3	2	1	0
I	Restore	0	0	0	0	h	V	r1	r0
I	Seek	0	0	0	1	h	V	r1	r0
I	Step	0	0	1	u	h	V	r1	r0
I	Step-in	0	1	0	u	h	V	r1	r0
I	Step-out	0	1	1	u	h	V	r1	r0
II	Read Sector	1	0	0	m	h	E	0	0
II	Write Sector	1	0	1	m	h	E	P	a0
III	Read Address	1	1	0	0	h	E	0	0
III	Read Track	1	1	1	0	h	E	0	0
III	Write Track	1	1	1	1	h	E	P	0
IV	Force Interrupt	1	1	0	1	13	12	11	10

Beschreibung der Variablen Bits:

h	Bedeutung
0	Motor einschalten
1	Motor ausschalten

V	Bedeutung
0	kein Verify
1	Verify

r1	r2	Steprate
0	0	2 ms
0	1	3 ms
1	0	5 ms
1	1	6 ms

u	Bedeutung
0	keine Neueinstellung der Trackregister
1	Neueinstellung der Trackregister

m	Bedeutung
0	einzelner Sektor
1	mehere Sektoren

a0	Bedeutung
0	schreibe normale DATA MARK
1	schreibe gelöschte DATA MARK

E	Bedeutung
0	keine Verzögerung
1	30 ms Verzögerung

P	Bedeutung
0	Precompensation ausgeschaltet
1	Precompensation eingeschaltet

l0 und l1 haben keine Bedeutung und sollten Null sein.
l2 löst einen Interrupt aus, wenn das Indexloch erreicht ist.
l3 bricht das laufende Kommando ab und löst einen Interrupt aus.
Wenn l0 bis l3 Null sind, wird das Kommando abgebrochen.

Programmierung und Bedeutung der einzelnen Kommandos:

An dieser Stelle möchten wir nochmal erwähnen, wie sie Driveselect und Sideselect ausführen können, denn dies wird nicht vom FDC gemacht, sondern vom Soundschip. Dieses läßt sich sehr einfach mit den »Extended Bios Kommandos *Ongibit()* und *Offgibit()* programmieren.

Sideselect 0: Offgibit(254)
Sideselect 1: Ongibit(1)

Driveselect A: Offgibit(253)
Driveselect B: Offgibit(251)

Driveselect A abschalten: Ongibit(2)
Driveselect B abschalten: Ongibit(4)

Um die Kommandos an den FDC zu schicken, müssen Sie in den »Supervisormode gehen. Der FDC wird über den DMA-Kontroller programmiert. Nach jedem Byte, das an den FDC geschickt wurde, sollte man einen Augenblick warten. Wir beschreiben die Program-

mierung allgemein, in dem wir sagen, in welcher Reihenfolge. Welche Speicherzelle geladen werden muß. Nach jedem Kommando sollte man das Bit 5 der Speicherzelle $FFFA01 testen und warten, bis es Null ist, denn denn ist das Kommando vom FDC ausgeführt worden.

Sie sollten vor jedem Kommando den DMA Puffer löschen. Dieses können sie folgendermaßen tuen:

$FF8606 = $90
$FF8606 = $190
$FF8606 = $90

Kommandos des Typ I:

Restore: Mit diesen Kommando können Sie die Diskettenstation auf Spur Null positionieren.

Ausführung:

$FF8606 = $80
Pause
$FF8604 = Kommando *Restore*

Seek: Diese Kommando erlaubt es, die Diskettenstation auf einer beliebigen Spur zu positionieren.

Ausführung:

$FF8606 = $86
Pause
$FF8604 = Spurnummer
Pause
$FF8606 = Kommando *Seek*

Step: Mit diesem Kommando können sie auf die nächste Spur wechseln. In welcher Richtung das geschieht, hängt von *Step-In* und *Step-Out* ab.

Ausführung:

$FF8606 = $80
Pause
$FF8604 = Kommando *Step*

Step-In: Dieses Kommando erlaubt es, auf die nächste Spur Richtung Track 76 zu wechseln.

Ausführung:

$FF8606 = $80
Pause
$FF8604 = Kommando *Step-In*

Step-Out: Mit diesem Kommando können Sie auf die nächste Spur in Richtung der Spur Null wechseln.

Ausführung:

$FF8606 = $80
Pause
$FF8604 = Kommando *Step-Out*

Kommandos des Typ II:

Read Sector: Mit diesem Kommando können Sie einen oder mehrere Sektoren lesen.

Ausführung:

$FF8609 = die unteren 8 Bit ihrer Pufferadresse
$FF860B = die Bits 8 bis 15 ihrer Pufferadresse
$FF860D = die Bits 16 bis 23 ihrer Pufferadresse
$FF8606 = $90
Pause
$FF8604 = Länge der Daten geteilt durch 512 (Der DMA transferiert immer 512 Byte Blöcke)
Pause
$FF8606 = $80
Pause
$FF8604 = Kommando *Read Sector*

Write Sector: Mit diesem Kommando können Sie einen oder mehere Sektoren schreiben.

Ausführung:

$FF8609 = die unteren 8 Bit ihrer Pufferadresse
$FF860B = die Bits 8 bis 15 ihrer Pufferadresse
$FF860D = die Bits 16 bis 23 ihrer Pufferadresse
$FF8606 = $90
Pause
$FF8604 = Länge der Daten geteilt durch 512 (Der DMA transferiert immer 512 Byte Blöcke)
Pause
$FF8606 = $180
Pause
$FF8604 = Kommando *Write Sector*

Kommandos des Typ III:

Read Address: Mit diesem Kommando können Sie feststellen, auf welcher Spur Sie sich befinden. Wenn eine Schecksumme nicht stimmt, wird ein CRC-Fehler generiert. Dieses Kommando liest den Header des nächsten Sektor auf dem Track ein. Die Tracknummer steht im Trackregister.

Der Header ist folgendermaßen aufgebaut:

Track, Seite, Sektor, Länge (wobei alle Werte Bytes sind)
Ausführung:

$FF8609 = die unteren 8 Bit ihrer Pufferadresse
$FF860B = die Bits 8 bis 15 ihrer Pufferadresse
$FF860D = die Bits 16 bis 23 ihrer Pufferadresse
$FF8606 = $80
Pause
$FF8604 = Kommando *Read Address*

Read Track: Dieses Kommando ist mit Vorsicht zu benutzen, denn die Informationen, die Read Track liefert, sind meistens fehlerhaft. Es scheinen nur die Sektorheader und die ID-Bytes zu stimmen.

Ausführung:

$FF8609 = die unteren 8 Bit ihrer Pufferadresse
$FF860B = die Bits 8 bis 15 ihrer Pufferadresse
$FF860D = die Bits 16 bis 23 ihrer Pufferadresse
$FF8606 = $90
Pause
$FF8604 = Länge der Daten geteilt durch 512 (Der DMA transferiert immer 512 Byte Blöcke)
Pause
$FF8606 = $80
Pause
$FF8604 = Kommando *Read Track*

Write Track: Mit diesem Kommando können Sie eine Spur formatieren. Der Track muß so gebaut sein, wie er beim Trackaufbau beschrieben wurde. Die Bytes F5 -FF dürfen nicht in den Sektoren stehen, da sie als Kommandos interpretiert werden. Statt CRC 1 und CRC 2 müssen Sie das Kommando FD in den Track einbauen, den der FDC generiert selber die CRC's. Das Kommando FE sagt, jetzt kommt der Sektorheader (Track, Seite, Sektor, Länge). Länge beschreibt die Länge der Sektoren:

Länge	Sektorlänge
0	128 Bytes
1	256 Bytes
2	512 Bytes
3	1024 Bytes

Das Kommando FB kündigt an, daß der Sektor folgt.

Ausführung:

$FF8609 = die unteren 8 Bit ihrer Pufferadresse
$FF860B = die Bits 8 bis 15 ihrer Pufferadresse
$FF860D = die Bits 16 bis 23 ihrer Pufferadresse
$FF8606 = $90
Pause
$FF8604 = Länge der Daten durch 512 geteilt (Der DMA transferiert immer 512 Byte Blöcke)
Pause
$FF8606 = $80
Pause
$FF8604 = Kommando *Write Track*

Kommandos des Typ IV:

Force Interrupt: Mit diesem Kommando können Sie laufende Kommandos abbrechen.

Bit	Bedeutung
10	keine Bedeutung
11	keine Bedeutung
12	Wenn Bit gesetzt, wird bei jedem Erreichen des Index-Loches ein Interrupt generiert (Abfrage durch Testen des Bit 5 der Speicherzelle FFFA01)
13	Wenn Bit gesetzt wird das Kommando abgebrochen und ein Interrupt generiert

Wenn die Bits 10 bis 13 Null sind, wird das laufende Kommando abgebrochen und kein Interrupt generiert.

Ausführung:

$FF8606 = $80
Pause
$FF8604 = Kommando *Force Interrupt*

Fehlermeldungen

F. werden von einigen Betriebssystemroutinen zurückgeliefert. Im allgemeinen zeigt eine Null an, daß kein Fehler aufgetreten ist.

Fehlermeldungen des XBIOS:

-1 = allgemeiner Fehler
-2 = Laufwerk ist nicht bereit.
-3 = unbekannte Funktionsnummer
-4 = An der seriellen Leitung (»V24-Schnittstelle) trat ein CRC-Fehler auf.
-5 = Die Parameter zu dieser Funktion sind nicht erlaubt.
-6 = Seek error; der »Floppy-controller meldet, daß die gewünschte Spur nicht gefunden werden konnte.
-7 = Der angegebene Bootsektor ist ungültig.
-8 = Der gewünschte Sektor wurde nicht gefunden.
-10 = Beim Schreiben auf das Laufwerk trat ein Fehler auf.
-11 = Beim Lesen vom Laufwerk trat ein Fehler auf.
-12 = allgemeiner Fehler
-13 = Der Schreibversuch wurde abgebrochen, weil die Diskette schreibgeschützt ist.
-14 = Die Diskette wurde gewechselt.
-15 = Das angesprochene Gerät existiert nicht.
-16 = Beim Überprüfen (»Verifizieren) wurde ein Fehler gefunden.
-17 = Das angesprochene Laufwerk enthält keine Diskette.

Fehlermeldungen des »GEMDOS:
-32 = unbekannte Funktionsnummer.
-33 = Datei wurde nicht gefunden.
-34 = Der gewünschte »Pfad existiert nicht.
-35 = Zuviele offene Dateien; es sind keine »Handles mehr übrig.
-36 = Der gewünschte Zugriff ist nicht möglich.
-37 = Diese »Handlenummer ist nicht bekannt.
-39 = Hierfür steht nicht genügend Speicher zur Verfügung.
-40 = Diese Speicherblockadresse ist nicht erlaubt, d.h. sie existiert entweder gar nicht oder gehört einem anderen »Prozeß.
-46 = Das gewünschte Laufwerk ist nicht bekannt.
-49 = Es existieren keine weiteren Dateien, die dem Suchstring genügen.
-64 = Ausserhalb der Reichweite (tritt bei der Funktion *Fseek()* auf).
-65 = Interner Fehler
-66 = Falsches Programmladeformat bei *Pexec()*.
-67 = *Mshrink()* mißglückt

Fenster
Siehe »Window und »Window Update

Fensternummer
siehe »Window Handle

File Selector Library
Zu »AES gehörende Routine.

Immer wieder benötigen Programme vom Benutzer einen Dateinamen, um Informationen auf Diskette zu speichern oder Dateien zu laden. »GEM stellt dafür eine recht komfortable Funktion zur Verfügung, die die Eingabe eines Dateinamens erleichtert: »*fsel_input()*.

Man übergibt der Funktion drei Zeiger: einen auf den »Pfadnamen, einen auf einen vorgegebenen »Dateinamen und einen Zeiger auf eine Variable für den Rückgabewert. »GEM erzeugt dann ein Dialogfenster (siehe die Abbildung 4 im Anhang A), in dem alle für Dateiwahl notwendigen Informationen bis auf eine stehen: wofür der Dateiname ausgewählt werden soll. Dies kann für den Benutzer unangenehme Folgen haben, wenn er nicht (mehr) weiß, ob er eine Datei zum Löschen oder zum Laden aussuchen soll...

Unter dem Ausdruck "Disk-Verzeichnis:" steht der aktuelle »Pfadname, beginnend mit der Laufwerksbezeichnung (oder mit "\" beginnend, dann wird das gerade aktuelle Laufwerk genommen) inklusive

einer Dateinamen-Erweiterung (".DOC", ".C" o. ä.). Darunter, in einem Fenster, stehen alle auf dem angegebenen Laufwerk in diesem Pfad gefundenen Dateien mit dem angegebenen Namen und der angegebenen »Extension (= Erweiterung; "*" und "?" sind für Namen und Erweiterung als sog. "Wildcards" bzw. "Joker" zugelassen, nicht aber für »Pfadnamen). Zusätzlich stehen Namen und Erweiterung noch im Titel-Balken des Fensters. An Dateinamen, die im Fenster keinen Platz mehr gefunden haben, kommt man heran, indem man (wie bei Fenstern üblich) die »Rollbalken und »Rollpfeile benutzt, um den Fensterinhalt zu scrollen.

Ordner werden mit einem Sonderkennzeichen bezeichnet. In Ordner kommt man hinein, indem man den Ordnernamen anklickt, und heraus, indem man das Schließfeld des Fensters anklickt (wie im »Desktop); befindet man sich bereits im Wurzelverzeichnis, so wird das »Directory der Diskette neu eingelesen (sofern man die Diskette gewechselt hat).

Der »Pfadnamen läßt sich von Hand ändern, indem man ihn mit der »Maus anklickt. Dann springt der Cursor in die Zeile, und der Name läßt sich editieren (mit der Taste "Cursor hoch" geht es auch). Der Fensterinhalt wird aber dem neuen »Pfadnamen erst angepaßt, wenn man mit der Maus in das »Fenster klickt. Drückt man <Return>, so verschwindet die »Dialogbox, denn das wirkt, als ob man den OK-Button angeklickt hätte... Existiert der »Pfadname nicht, so zeigt das »Fenster nur leere Zeilen! Will man ein anderes Laufwerk auswählen, so muß (wie oben beschrieben) der »Pfadname editiert werden.

Um nun eine Datei auszuwählen, kann man entweder einen Namen doppelt anklicken oder den Namen einmal anklicken (er wird dann invertiert und erscheint unter "Auswahl:") und dann <Return> drükken (bzw. den OK-Button anklicken) oder unter "Auswahl:" einen neuen Namen eintragen und "OK" auswählen. Will man nichts wählen, so klicke man "ABBRUCH" an.

Filesystem

Ein F. beschreibt die Struktur eines Massenspeichers wie z.B. einer Festplatte oder einer Floppy-Disk. Das einfachste F. besteht natürlich darin, einfach alle Files hintereinander aufzuführen. Dann verstreicht allerdings die meiste Zeit am Rechner damit, die richtigen Files wiederzufinden. Deshalb werden für größere Rechner mehr oder weniger strukturierte F.e eingesetzt. Im Atari ST wird ein sogenanntes »hierarchisches Filesystem benutzt, wie es auch von MS/DOS und UNIX™ her bekannt ist.

Floppdiskcontroller

Sehen Sie dazu unter »FDC nach.

Font

Amer.: Zeichensatz. Das Betriebssystem des Atari ST stellt drei Zeichensätze zur Verfügung, die sich nur in ihrer Größe unterscheiden. In einem F. haben die Zeichen eine Größe von 16*8 Punkten. Dieser F. wird im hochauflösenden Modus (640*400 Punkte) benutzt. Die Zeichen im zweiten F. haben eine Größe von 8*8 Punkten und werden in den Farbmodi verwendet. Im dritten F. sind die Zeichen 6*6 Punkte groß und werden im hochauflösenden Modus für die Beschriftung von »Icons benutzt.

Jeder der drei F.s stellt dieselben Zeichen zur Verfügung. Die Zuordnung der Zeichen zu den Codes entspricht dem IBM-ASCII Format, wobei alle 256 möglichen Codes benutzt werden (zumindest bei Benutzung der »BIOS Routine 3 *Conout* mit Devicenummer 5). Diese Zuordnung kann der folgenden Tabelle entnommen werden.

	$.0	$.1	$.2	$.3	$.4	$.5	$.6	$.7	$.8	$.9	$.A	$.B	$.C	$.D	$.E	$.F
$0.		↑	↓	→	←	▒	▓	▒	✓	◷	♠	♪	♀F	♂R	⁄	⁄
$1.	0	!	2	3	4	5	6	7	8	9	a	E5	e	n	⁄	⁄
$2.		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
$3.	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
$4.	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
$5.	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
$6.	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
$7.	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	△
$8.	Ç	ü	é	â	ä	à	å	ç	ê	ë	è	ï	î	ì	Ä	Å
$9.	É	æ	Æ	ô	ö	ò	û	ù	ÿ	Ö	Ü	¢	£	¥	₧	ƒ
$A.	á	í	ó	ú	ñ	Ñ	ª	º	¿	⌐	¬	½	¼	¡	«	»
$B.	ã	õ	Ø	ø	œ	Œ	Ã	Õ	Ø	¦	¦	¦	¦	¦	®	™
$C.	ij	IJ	ж	⅀	⅄	⅂	⅃	⅁	⅂	⅃	⅃	⅃	⅃	⅃	⅃	⅃
$D.	⅁	⅃	⅃	⅃	⅃	⅃	⅃	⅃	⅃	⅃	⅃	⅃	⅃	⅃	∧	∞
$E.	α	β	Γ	π	Σ	σ	µ	τ	Φ	Θ	Ω	δ	φ	φ	Є	∩
$F.	≡	±	≥	≤	⌠	⌡	÷	≈	°	•	•	√	n	2	3	¯

F.s sind nachladbar und müssen einem bestimmten Aufbau genügen. Jedem Font steht ein Header voran, der alle wesentlichen Informationen über diesen F. enthält. Dieser Header ist 88 Bytes lang und hat folgenden Aufbau:

$00 = 0 Font Identifier. Diese Zahl gibt an, um welchen F. es sich handelt.

$02 = 2 Fontgröße in *Punkt* (Maß der Satztechnik).

$04 = 4 Der Fontname als normale ASCII-Zeichenkette, die mit einem Nullbyte beendet wird.

$24 = 36 ASCII-Wert des ersten darstellbaren Zeichens.

$26 = 38 ASCII-Wert des letzten darstellbaren Zeichens.
$28 = 40 Relativer Abstand vom Top zur Base Line.
$2A = 42 Relativer Abstand vom Ascent zur Base Line.
$2C = 44 Relativer Abstand vom Half zur Base Line.
$2E = 46 Relativer Abstand vom Descent zur Base Line.
$30 = 48 Relativer Abstand vom Bottom zur Base Line.
$32 = 50 Breite des breitesten Zeichens im Font.
$34 = 52 Breite des breitesten Feldes. Damit zwei Zeichen getrennt dargestellt werden, muß diese mindestens um Eins größer sein als der Wert in $32.
$36 = 54 Offset, um den die Bottom Line gegenüber der Base Line beim Kursivstellen nach links verschoben wird.
$38 = 56 Offset für die Top Line.
$3A = 58 Thickening. Angabe der Strichbreite für Fettdruck.
$3C = 60 Underline. Legt die Höhe des Striches gegenüber der Base Line fest.
$3E = 62 Lightening mask. Mit dieser Maske werden Zeichen UND-verknüpft wenn, sie *light* dargestellt werden sollen.
$40 = 64 Skewing mask. Diese Maske wird für kursiv und *light* dargestellte Zeichen benutzt.
$42 = 66 Flagbyte: Bit 1 wird gesetzt, wenn es sich um den Systemfont handelt.
Bit 2 wird gesetzt, wenn die Bytes im Motorola Format (high/low) im Speicher stehen.
Bit 3 wird gesetzt, wenn alle Zeichen im Font gleich breit sind.
$44 = 68 Zeiger auf die Horizontal Offset Table.
$48 = 72 Zeiger auf die Character Offset Table.
$4C = 76 Zeiger auf die Fontdaten.
$50 = 80 Form width. Gibt die Gesamtbreite aller Zeichen an und stellt somit gleichzeitig den Offset zur nächsten Scanline dar.
$52 = 82 Form height. Gibt die Anzahl der Scanlines an.
$54 = 84 Enthält einen Zeiger auf den nächsten Font.

Die eigentlichen Fontdaten sind so abgelegt, daß kein Bit verschwendet wird. Angenommen, das erste Zeichen sei 10 Pixel breit und das zweite acht. Der Zeiger in $4C zeigt dann auf das Byte, das die oberste Zeile des ersten Zeichens definiert. Da dieses jedoch 10 Pixel breit ist, werden die beiden restlichen Pixel im nächsten Byte untergebracht. Die Bits 5 bis 0 werden dann von den ersten 6 Pixeln des zweiten Zeichens beansprucht, welches damit jedoch ebenfalls nicht auskommt und somit ins dritte Byte hineinragt. Dies setzt sich durch den gesamten Font fort. Die zweite Scanline (Pixelreihe für alle Zeichen) beginnt dann wieder auf einer geraden Adresse.

Damit ein Zeichen auch gefunden werden kann, muß bekannt sein, ab welcher Pixelposition es beginnt und wo es endet. Für diesen

Zweck gibt es die sogenannte Character Offset Table. Diese enthält für jedes Zeichen den Offset vom Beginn einer Scanline bis zum ersten Pixel des Zeichens.

Form

Eine F. ist entweder das englische Wort für "Formular" (siehe unter »Dialogboxen) oder ein rechteckiger Ausschnitt aus dem »Bildschirm oder dem Speicher. Siehe »Raster

Form_alert

zur »Form Library gehörende Routine.

```
$34 = 52        fo_aexbttn = form_alert(fo_adefbttn, fo_astring)
word            fo_aexbttn                fo_adefbttn;
byte *                                    fo_astring;
```

Diese Routine stellt eine »Alarmbox dar. Diese ist einfacher zu erzeugen und zu benutzen als eine »Dialogbox, da kein »Objektbaum konstruiert werden muß (dies macht die Routine selbst). Außerdem wird der Hintergrund, der durch die »Alarmbox übermalt wird, zwischengespeichert und bei Verschwinden der Box wieder restauriert. Beispiel:

```
exit = form_alert(1, "[1][Sie lesen da gerade ein\
Buch!¦Stimmt's?][ Ja ¦ Nein ¦ Vielleicht ]");
```

Das Beispiel ist natürlich für C-Programmierer: der C-Compiler ersetzt beim Compilieren den String als Parameter durch seine Adresse! Mit Aufruf der Routine geschieht folgendes:

Eine »Alarmbox erscheint mit einem »Icon (ein Ausrufezeichen). In der Box steht ein zweizeiliger Text (das Zeichen "¦" ist ein Zeilentrenner). Ferner befinden sich unter dem Text drei »Buttons: der erste hat eine fette Umrahmung (kann also mit <Return> angewählt werden), die beiden anderen befinden sich rechts daneben (und müssen zur Wahl angeklickt werden). Der Text der »Buttons wird (bei mehreren) ebenfalls durch das Zeichen "¦" voneinander getrennt (nicht durch [|!). Zu beachten ist ebenfalls, daß die drei Buttons nebeneinander weniger Platz einnehmen dürfen als die längste Zeile! Das Wählen irgendeines Buttons führt zum Verschwinden der Box.

Die Nummer im String in rechteckigen Klammern gibt an, welches »Icon in der Box stehen soll:

0 - kein »Icon
1 - Ausrufezeichen
2 - Fragezeichen
3 - Stopzeichen

Einige Einschränkungen gibt es:

-Die »Alarmbox darf nicht größer sein als ein Viertel des Bildschirms
-Eine Textzeile darf nicht mehr als 40 Zeichen lang sein
-Maximal drei »Buttons sind erlaubt

Eingabeparameter:

word fo_adefbttn Bestimmt die Nummer des "Default"-Buttons. Das ist der Knopf, der fett dargestellt wird und auch durch drücken von <Return> ausgewählt werden kann. Wobei gilt:

0 = kein "Default"-Button
1 = erster Button
2 = zweiter Button
3 = dritter Button

byte *fo_astring Adresse des Strings, der die nötigen Angaben zur Darstellung der Box enthält. Aufbau des Strings: s. o.

```
contrl(0)   =  52
contrl(1)   =   1
contrl(3)   =   1

int_in(0)   =  fo_adefbttn

addr_in(0)  =  fo_astring
```

Ausgabeparameter:

word fo_aexbttn Die Nummer des »Buttons, den der Benutzer gewählt hat:

1 - erster »Button in 'fo_astring'
2 - zweiter »Button
3 - dritter »Button

```
contrl(2)   =   1
```

```
contrl(4)   =  0

int_out(0)  =  fo_aexbttn
```

Form_center

zur »Form Library gehörende Routine.

```
$36 = 54        fo_cresvd = form_center(fo_ctree, &fo_cx, &fo_cy
word            fo_cresvd;
word *                                  fo_ctree,  fo_cx,  fo_cy;
                                        &fo_cw, &fo_ch)
word *                                   fo_cw,  fo_ch;
```

Mit dieser Routine wird eine »Dialogbox auf dem »Bildschirm zentriert. Dazu muß (wieder einmal) die Adresse des »Objektbaums übergeben werden. Im Wurzelobjekt werden dann die X- und Y-Koordinaten entsprechend der Breite und Höhe des Objekts berechnet und angepaßt. Außerdem werden die Rückgabewerte der Funktion benötigt, um mittels »*form_dial()* den Bildschirmausschnitt reservieren zu können (mehr dazu bei der Erläuterung zu den »Dialogboxen).

Eingabeparameter:

```
word *fo_ctree      Die Adresse des »Objektbaums der »Dialogbox

word *fo_cx
word *fo_cy
word *fo_cw         Die Adressen der Variablen für Rückgabewerte
word *fo_ch         (s. u.)

contrl(0)   =  54
contrl(1)   =   0
contrl(3)   =   1

addr_in(0)  =  fo_ctree
```

Ausgabeparameter:

```
word fo_cresvd      Ein nutzloser Rückgabewert - er ist immer
                    gleich Eins

word fo_cx          Die X-Koordinate der »Dialogbox bei zentrier-
                    ter Darstellung

word fo_cy          Die Y-Koordinate der »Dialogbox bei zentrier-
                    ter Darstellung
```

```
word fo_cw          Die Breite der »Dialogbox

word fo_ch          Die Höhe der »Dialogbox

contrl(2)   =   5
contrl(4)   =   0

int_out(0)  =  fo_cresvd
int_out(1)  =  fo_cx
int_out(2)  =  fo_cy
int_out(3)  =  fo_cw
int_out(4)  =  fo_ch
```

Form_dial

zur »Form Library gehörende Routine.

```
$33 = 51        fo_direturn = form_dial(fo_diflag,
word            fo_direturn,            fo_diflag;
                                        fo_dilitlx, fo_dilitly,
word                                    fo_dilitlx, fo_dilitly;
                                        fo_dilitlw, fo_dilitlh,
word                                    fo_dilitlw, fo_dilitlh;
                                        fo_dibigx, fo_dibigy,
word                                    fo_dibigx, fo_dibigy;
                                        fo_dibigw, fo_dibigy)
word                                    fo_dibigw, fo_dibigy;
```

Diese Routine erfüllt vier verschiedenen Zwecke: sie reserviert Platz auf dem »Bildschirm für eine »Dialogbox, sie gibt den reservierten Platz wieder frei, sie zeichnet ein sich vergößerndes oder aber ein sich verkleinerndes Rechteck.

Wozu muß Platz reserviert werden? Nun, wenn man eine »Dialogbox mit der Routine »*objc_draw()* auf dem »Bildschirm zeichnet, so werden damit »Fenster, Zeichnungen, andere »Objekte oder zumindest der Desktop-Hintergrund überdeckt. Platz reservieren heißt, »AES mitzuteilen, daß in dem reservierten Gebiet keine Veränderung mehr stattfinden darf: sonst könnte beispielsweise ein »Drop-Down-Menü die »Dialogbox (teilweise) zerstören. Den reservierten Platz wieder freigeben bedeutet, daß »AES dieses Gebiet wieder benutzen darf **und vor allem** daß »AES der »Application in diesem Gebiet mitteilt (siehe »Message), daß die dort befindlichen »Fenster neu gezeichnet werden müssen. Das Neuzeichnen des Desktop-Hintergrunds übernimmt »AES selbst.

Bei der Erklärung von »*wind_set()* hatten wir erwähnt, daß es möglich ist, einen neuen Desktop-Hintergrund zu kreieren. Der unter 'WF_NEWDESK' erwähnte Funktionsaufruf alleine schafft nur dann

einen neuen Hintergrund, wenn »AES angewiesen wird, den Hintergrund neu zu zeichnen. Mit *form_dial()* ist das möglich, wenn man den gesamten Hintergrund einfach freigibt (als ob er zerstört worden wäre...).
Ferner kann mit *form_dial()* das Neuzeichnen eines bestimmten Bereichs auf dem »Bildschirm erzwungen werden.

Eingabeparameter:

word fo_diflag — Mit dem Wert dieses Parameters bestimmt man, welche der oben erwähnten vier Funktionen ausgelöst werden:

0 - FMD_START	reserviert Bildschirmbereich
1 - FMD_GROW	ruft »*graf_growbox()* auf, um ein sich vergrößerndes Rechteck zu zeichnen
2 - FMD_SHRINK	ruft *graf_shrinkbox()* auf, um ein sich verkleinerndes Rechteck zu zeichnen
3 - FMD_FINISH	gibt Bildschirmbereich wieder frei und veranlaßt »AES, den Hintergrund neu zu zeichnen und der »Application eine 'REDRAW'-Message zu schicken

Die symbolische Namen (FMD_START etc.) sind der Datei "GEMDEFS.H" entnommen.

word fo_dilitx — Die X-Koordinate des kleinsten Rechtecks für FMD_GROW oder FMD_SHRINK, sonst ohne Belang

word fo_dility — Die Y-Koordinate des kleinsten Rechtecks für FMD_GROW oder FMD_SHRINK, sonst ohne Belang

word fo_dilitw — Die Breite des kleinsten Rechtecks für FMD_GROW oder FMD_SHRINK, sonst ohne Belang

word fo_dilith — Die Höhe des kleinsten Rechtecks für FMD_GROW oder FMD_SHRINK, sonst ohne Belang

word fo_dibigx — Die X-Koordinate des größten Rechtecks für FMD_GROW oder FMD_SHRINK; die X-Koordinate des reservierten Bereichs für FMD_START oder FMD_FINISH

word fo_dibigy — Die Y-Koordinate des größten Rechtecks für FMD_GROW oder FMD_SHRINK; die Y-Koordinate des reservierten Bereichs für FMD_START oder FMD_FINISH

word fo_dibigw — Die Breite des größten Rechtecks für FMD_GROW oder FMD_SHRINK; die Breite des reservierten Bereichs für FMD_START oder FMD_FINISH

word fo_dibigh — Die Höhe des größten Rechtecks für FMD_GROW oder FMD_SHRINK; die Höhe des reservierten Bereichs für FMD_START oder FMD_FINISH

```
contrl(0)   =  51
contrl(1)   =   9
contrl(3)   =   0

int_in(0)   =  fo_dilitx
int_in(1)   =  fo_dility
int_in(2)   =  fo_dilitw
int_in(3)   =  fo_dilith
int_in(4)   =  fo_dibigx
int_in(5)   =  fo_dibigy
int_in(6)   =  fo_dibigw
int_in(7)   =  fo_dibigh
```

Ausgabeparameter:

word fo_direturn 0 = Ein Fehler ist passiert, n = alles Ok.

```
contrl(2)   =   1
contrl(4)   =   0

int_out(0)  =  fo_direturn
```

Form_do

zur »Form Library gehörende Routine.

```
$32 = 50        fo_doreturn = form_do(fo_dotree, fo_dostartob)
word            fo_doreturn,                     fo_dostartob;
word *                                  fo_dotree;
```

Diese Routine ist der Schlüssel zu den »Dialogboxen! Sie übernimmt die komplette Überwachung der Interaktion des Benutzers (und seiner Maus) mit der »Dialogbox. Dies geschieht solange, bis der Benutzer ein »Objekt aktiviert, welches als "EXIT"-Objekt gekennzeichnet wurde (näheres siehe »Dialogboxen!). Der Wert dieses »Objekts wird zurückgegeben. Die »Dialogbox darf übrigens auch

editierbare Textfelder enthalten. Ihre Größe kann maximal die Größe des Desktop-Fensters sein.

Eingabeparameter:

word *fo_dotree Adresse des »Objektbaums, der die Daten der Dialogbox enthält

word fo_dostartob Die Indexnummer des (editierbaren) Textfeldes, in dem zu Anfang der Cursor stehen soll. Falls kein editierbares Textfeld vorhanden ist, **muß eine NULL** übergeben werden (und nicht -1, wie oft fälschlich behauptet)

```
contrl(0)   =  50
contrl(1)   =   1
contrl(3)   =   1

int_in(0)   =  fo_dostartob

addr_in(0)  =  fo_dotree
```

Ausgabeparameter:

word fo_doreturn Die Indexnummer des »Objekts, dessen Aktivierung zum Beenden des Dialogs geführt hat

```
contrl(2)   =   1
contrl(4)   =   0

int_out(0)  =  fo_doreturn
```

Form_error

zur »Form Library gehörende Routine.

```
$35 = 53        fo_eexbttn = form_error(fo_enum)
word            fo_eexbttn,             fo_enum;
```

Mit dieser Routine lassen sich die »Alarmboxen aufrufen, die das Betriebssystem des Atari bei Diskettenfehlern verwendet (z. B.: "Diese Diskette ist schreibgeschützt..."). Dabei muß eine der bei »GEMDOS erläuterten Fehlernummern übergeben werden.

Eingabeparameter:

word fo_enum Die Fehlernummer des aufgetretenen Fehlers (siehe »GEMDOS)

```
contrl(0)   =  53
contrl(1)   =   1
contrl(3)   =   0
```

int_in(0) = fo_enum

Ausgabeparameter:

word fo_eexbttn Die Nummer des »Buttons, der vom Benutzer ausgewählt wurde

contrl(2) = 1
contrl(4) = 0

int_out(0) = fo_eexbttn

Form Library

Zur F. gehören eine Reihe von Funktionen, die zur Darstellung von »Dialog- und »Alarmboxen dienen ("Form" heißt auf deutsch soviel wie "Formular"). Zur Erzeugung von »Dialogboxen gibt es ein eigenes Stichwort.

Folgende Routinen gehören zur F.:

-»*form_alert()* Dient der Darstellung von »Alarmboxen

-»*form_center()* Damit lassen sich »Dialogboxen auf dem Bildschirm zentrieren

-»*form_dial()* Mehrzweck-Funktion, reserviert Platz auf dem »Bildschirm oder zeichnet ein sich vergrößerndes oder sich verkleinerndes Rechteck

-»*form_do()* Überwacht die Interaktion des Benutzers mit der »Dialogbox

-»*form_error()* »Alarmbox für »DOS-Fehler

Unter dem Stichwort »Dialogbox finden Sie einen Teil der Routinen im Zusammenhang erklärt.

Fsel_input

Einzige zur »File Selector Library gehörende Routine.

```
$5A = 90        fs_ireturn = fsel_input(fs_iinpath, fs_iinsel,
word            fs_ireturn;
word *                                  fs_iinpath, fs_iinsel;
                                        &fs_iexbutton)
word *                                   fs_iexbutton;
```

Was es mit einer "File Selector Box" zur Auswahl einer Datei auf

Diskette auf sich hat, können Sie unter dem Stichwort »File Selector Library nachlesen.

Eingabeparameter:

byte *fs_iinpath	Zeiger auf einen 64 Byte langen String, in dem der Pfadname steht. Hat der Benutzer diesen »Pfadnamen editiert, so steht die geänderte Version nach Rückkehr der Funktion in diesem String.
byte *fs_iinsel	Zeiger auf einen 16 Byte langen String, in dem ein Default-Dateiname steht, der unter "Auswahl" angezeigt wird. Ist der String leer (erstes Zeichen eine NULL), so steht da nichts.
word *fs_iexbutton	Zeiger auf eine Variable für Rückgabewert (s. u).

```
contrl(0)   =  90
contrl(1)   =   0
contrl(3)   =   2

addr_in(0)  =  fs_iinpath

addr_in(1)  =  fs_iinsel
```

Ausgabeparameter:

word fs_ireturn	0 = ein Fehler ist passiert, n (irgendeine positive Zahl) = alles in Ordnung
word fs_iexbutton	0 = Benutzer hat "ABBRUCH" angeklickt 1 = Benutzer hat "OK" angeklickt

```
contrl(2)   =   2
contrl(4)   =   0

int_out(0)  =  fs_ireturn
int_out(1)  =  fs_iexbutton
```

Function-Code

Die CPU »MC 68000 besitzt drei Pins, an denen ihr interner Zustand abgelesen werden kann. Der daraus resultierende Wert wird z.B. bei einem »Bus-Error mit auf den Stack gelegt und kann dann vom Programmierer ausgewertet werden. Diese Codes haben folgende Bedeutung:

FC2	FC1	FC0	Interner Zustand der CPU
0	0	0	Diese Kombination wird nicht benutzt.
0	0	1	Es werden User Daten verarbeitet (»Usermode).
0	1	0	Es wird ein User Befehl geholt (»Usermode).
0	1	1	Diese Kombination wird nicht benutzt.
1	0	0	Diese Kombination wird nicht benutzt.
1	0	1	Es werden Supervisor Daten verarbeitet.
1	1	0	Es wird ein Supervisorbefehl geholt.
1	1	1	Diese Kombination wird nicht benutzt.

Funktionstasten

Als F. bezeichnet man Sondertasten, deren Verwendungszweck erst in Programmen bestimmt wird (über die also Funktionen ausgelöst werden). Die F. finden Sie beim Atari über der Schreibmaschinentastatur in der hübschen (aber unpraktischen) 45-Grad-Form. Die Abfrage, ob eine der gedrückten Tasten eine F. war, geschieht mittels einer »BIOS- oder »GEMDOS-Funktion. Siehe auch »Scancode

GDP

siehe unter »Generalized Drawing Primitive

GDOS

Abkürzung für Graphics Device Operating System, zu deutsch "Grafisches Gerätetreiber-System". GDOS ist ein Programm ("GDOS.PRG"), welches Gerätetreiber für »VDI zur Verfügung stellt. GDOS ist ein residentes Programm, welches zu Anfang in einem AUTO-Ordner gestartet werden muß. Es verbleibt dann im Speicher. Eine spätere Installation von GDOS nützt nichts!

GDOS ist notwendig, wenn Sie
- »NDC-Koordinaten in Ihrem Programm verwenden wollen, oder
- andere »Fonts (Zeichensätze) laden möchten.
- oder andere Gerätetreiber (z. B. Druckertreiber) benutzen möchten

GDOS benötigt eine Datei namens "ASSIGN.SYS" auf der Diskette. In dieser Datei stehen eine Anzahl von Gerätetreibern (als Liste) und »Fonts. Die Treiber bzw. »Fonts können später von Ihrem Programm nachgeladen werden. ACHTUNG! Das letzte Zeichen in ASSIGN.-SYS **muß** $1A sein (Control Z)!

Im folgenden finden Sie ein Beispiel für die Datei "ASSIGN.SYS". Kommentare sind durch ";" abgetrennt worden. Was hinter dem Semikolon steht, wird von GDOS ignoriert. Namen mit der »Extension ".SYS" bezeichnen Gerätetreiber, mit ".FNT" enden Zeichensätze:

```
path = a:\zeichen\      ;Dies ist der Suchpfad für die »Fonts
;                        Diese werden später aus einem Ordner
;                        namens "Zeichen" geholt
01p screen.sys          ;Voreinstellung für Bildschirm
                        ;Diese »Fonts können später mit der Rou-
                        ;tine »vst_load_fonts() nachgeladen wer-
                        ;den
XYZPHS08.FNT            ;Zeichensätze
XYZPSS16.FNT
XYZPDS32.FNT
XYZPHD08.FNT
XYZPSD16.FNT
XYZPDD32.FNT
;
;
02p screen.sys          ;Niedrige Bildschirmauflösung 320x200
;
XYZPHS08.FNT            ;Zeichensätze
XYZPSS16.FNT
XYZPDS32.FNT
```

```
XYZPHD08.FNT
XYZPSD16.FNT
XYZPDD32.FNT
;
03p screen.sys              ;Mittlere Bildschirmauflösung 640x200
;
XYZPHS08.FNT
XYZPSS16.FNT
XYZPDS32.FNT
XYZPHD08.FNT
XYZPSD16.FNT
XYZPDD32.FNT
;
04p screen.sys              ;Hohe Auflösung 640x400
;
XYZPHS08.FNT
XYZPSS16.FNT
XYZPDS32.FNT
XYZPHD08.FNT
XYZPSD16.FNT
XYZPDD32.FNT
;
21 fx80.sys                 ;Druckertreiber für FX80-Drucker
31 META.SYS                 ;Treiber für Metafiles
```

Es gibt mehrere Gerätetypen: »Bildschirm, Plotter, Drucker etc. Vor dem Gerätetyp steht eine Gerätenummer. Folgende Identifikationsnummern sind vorgesehen:

Gerätetyp	Gerätenummer(n)
Bildschirm	1-10
Plotter	11-20
Drucker	21-30
Metafile	31-40
Kamera	41-50
Grafiktablett	51-60

Nach dem Gerätetyp "Bildschirm" (screen.sys 1-4) stehen die jeweils für die entsprechende Auflösung gebrauchten Zeichensätze. Die meisten Softwarehersteller schreiben leider für ihre Programme ganz eigene Druckertreiber, anstatt Standard-Druckertreiber für »VDI zu benutzen oder zu entwerfen. Auch für andere Geräte existieren unseres Wissens (noch) keine Treiber, so daß die hier gegebenen Informationen nur Anregung sein können. Das Schreiben eigener Gerätetreiber setzt allerdings exakte Kenntnisse über GDOS voraus, die (momentan) niemand hat, den wir kennen...

Es gibt zwei Versionen von GDOS. Die ältere Version hat einen Fehler, der verhindert, daß weitere Zeichensätze nachgeladen werden können. Die alte Version meldet sich nach dem Start mit "GEMVDI resident", die neue mit "Atari GDOS Version 1.1".

GEM

GEM ist eine Warenzeichen von Digital Research. Es ist die Abkürzung von Graphics Environment Manager, zu deutsch soviel wie "Verwalter für Grafische Gestaltung".

GEM (wird ausgesprochen wie "Dschäm" und bedeutet soviel wie Edelstein oder Prachtstück) ist ein für eine Reihe von Mikrocomputern entwickeltes »Betriebssystem, das sehr viele verschiedene Routinen zur Gestaltung einer grafischen »Benutzeroberfläche bereithält. Die grafische Benutzerschnittstelle läßt sich kennzeichnen durch die Verwendung der »Maus als Eingabegerät. Die »Maus ist ein kleiner Kasten, der beim Verschieben auf dem Tisch einen Cursor (z. B. einen Pfeil) synchron dazu bewegt. Kommandos lassen sich ausführen, indem der »Mauszeiger auf den entsprechenden Befehl gerichtet wird und eine Taste auf der »Maus ein- oder zweimal gedrückt wird. Die Kommandos können durch kleine Bildchen (genannt »Icons) oder durch Text symbolisiert werden. Letztere Darstellung findet man in den sog. »Drop-Down-Menüs. Diese »Menüs werden am oberen Bildschirmrand als eine Textzeile angezeigt, bestehend aus einer Reihe von Wörtern (wird als "Menüzeile bezeichnet). Berührt die »Maus eines dieser Wörter, so wird dieses invertiert (mit weißer Schrift auf schwarzem Grund dargestellt), und unter dem Wort erscheint ein Kasten, in dem sich Unterpunkte befinden, die Kommandos darstellen. Eines dieser Kommandos kann ausgeführt werden, indem man das zugehörige Wort im Kasten mit dem »Mauszeiger berührt (jedes vom Mauspfeil berührte Wort wird ebenfalls invertiert). Drückt man nun die linke Maustaste, so wird das Kommando ausgeführt. Dies ist eine sehr einfache Methode, um mehrere Möglichkeiten zur Auswahl anzubieten. Auch »Dialogboxen werden von GEM unterstützt. Dies sind grafisch dargestellte Kästen, die verschiedene »Objekte beinhalten, die wiederum durch "Anklicken" mit der Maus ausgewählt werden können. Auch Text kann in entsprechende Textfelder eingetragen werden. Für nähere Informationen schlagen Sie bitte die in diesem Absatz erwähnten Stichwörter nach!

GEM besteht aus folgenden Teilen:

- »VDI = Virtual Device Interface (= virtueller Gerätetreiber) Hierbei handelt es sich um eine Fülle von Routinen zum Erstellen von Grafik. Siehe »VDI

- »AES = Application Environment Services (= Service zum Gestalten einer Umgebung für »Anwendungen) Dies ist eine Sammlung von Routinen und Programmen, die von »Applications (= »Anwendungen) genutzt werden können. Siehe »AES

- »Desktop (= Schreibtisch)
  Dies ist die grafische »Benutzeroberfläche, die Sie beim Einschalten des Atari erwartet. Von ihr können Programme gestartet werden etc. Siehe »Desktop

- »GDOS = Graphics Device Operating System (= grafischer Gerätetreiber)
  Dies ist ein Programmteil, der dafür verantwortlich ist, daß es Ihrem Programm gleichgültig sein kann, ob ein »Bildschirm, ein Plotter, ein Drucker oder ein sonstiges Ein- oder Ausgabegerät an Ihren Rechner angeschlossen ist. Allerdings gehören dazu noch verschiedene Gerätetreiber. Siehe »GDOS

Für den Computer-Benutzer (auf Neudeutsch: User) bedeutet GEM eine einheitliche Benutzeroberfläche für die verschiedenartigsten Programme. Für den Programmierer bedeutet GEM, grafik-orientierte Programme für einen Rechner zu schreiben und dieses dann ohne viel Aufwand auf einen anderen Rechner zu übertragen. Allerdings muß dazu erwähnt werden, daß die Programmierung von GEM nichts für Anfänger ist!

Falls Sie genauer in die geheimnisvolle Welt von GEM einsteigen wollen, so stellt sich folgendes Problem: Um einen Teil von GEM zu begreifen, muß das ganze GEM verstanden worden sein. Um das ganze GEM zu verstehen, muß man seine Teile begriffen haben... Und da beißt sich die Katze in den Schwanz! Der beste Weg, um einen Einstieg in GEM zu finden, ist noch nicht gefunden worden. So können wir nur ein paar Empfehlungen geben:

- Lesen Sie sich die Stichworte GEM, »VDI und »AES durch.

- Falls Sie etwas nicht verstehen, machen Sie sich nichts daraus. Das ist normal (aber kein Dauerzustand!?).

- Beschäftigen Sie sich als erstes mit »VDI. Grundlegend für die Verwendung der »VDI-Befehle ist die Routine »*v_opnvwk()*. Nicht alle Parameter für diese Funktion müssen verstanden worden sein!

- Sehen Sie sich die »VDI-Befehle zu den »Generalized Drawing Primitives an. Probieren Sie diese Befehle aus (mit C, Basic oder in Assembler). Probieren geht **immer** über Studieren!

- Wenn Sie die grundlegenden »VDI-Befehle benutzen können, sind Sie einen gewaltigen Schritt weiter. Versuchen Sie sich an ausgefalleneren Befehlen.

- Sehen Sie sich »AES an. Von »AES wiederum ist zunächst die »Application Library interessant (und dort eigentlich nur die Routinen »*appl_init()* und *appl_exit()*)

- Der nächste Schritt besteht darin, das Stichwort »Ereignis (»Event) verstehen zu lernen. Lesen Sie dazu unter »Event Library nach, dann die dazugehörigen Routinen. Von zentraler Bedeutung ist »*evnt_multi()*, aber um das zu verstehen, müssen Sie erst alle anderen in der »Event Library existierenden Routinen begriffen haben!

- Schreiben Sie ein Programm, das eine »Anwendung anmeldet (mittels »*appl_init()*), eine virtuelle Arbeitsstation öffnet (mit »*v_opnvwk()*), eine »*evnt_multi()*-Schleife besitzt und Ihnen in dieser Schleife die eingetretenen Ereignisse mitteilt.

- Jetzt können Sie sich an »Drop-Down-Menüs wagen. Sehen Sie sich dazu die Stichwörter »Drop-Down-Menü und »Menu Library an.

- Da das so schön einfach ging, können Sie sich mit »Fenstern beschäftigen. Sehen Sie sich die Stichworte »Graphic Library und »Window Library an (samt den Routinen). Falls Sie bei den »Windows etwas nicht verstehen, machen Sie einen Abstecher zu »Event.

- Nachdem Sie nun »Fenster öffnen und schließen können, zeichnen Sie mit »VDI-Funktionen etwas in die »Fenster hinein. Vergessen Sie das »Clipping nicht (siehe auch »*vs_clip()*)!

- Weiter geht es mit mit »Window-Update. Wenn Sie das beherrschen, bietet die Fensterbenutzung keine Rätsel mehr.

- Jetzt können Sie sich den »Dialogboxen zuwenden. Sehen Sie sich dazu **vor allem** das Stichwort »Objekt an, dann erst das Stichwort »Dialogboxen. Von dort werden Sie (wie zuvor auch) an andere wichtige Stellen verwiesen.

- Fangen Sie wieder von vorne an! Sie werden jetzt alles mit neuen Augen lesen. Und zwar nicht nur, weil Sie etwas älter geworden sind...

Bei den meisten Stichwörtern finden Sie Verweise auf weitere Einträge. Manchmal kann es Ihnen passieren, daß Sie sich im Kreis drehen. Glauben Sie uns: das ist keine böswillige Absicht, sondern eine unumgängliche Notwendigkeit. Leider.

Der wichtigste Rat lautet:
Nur keine Panik! (Don't Panic!)

Viel Spaß!

GEMBIND.H

Das ist eine von Atari mit dem Entwicklungspaket vertriebene Datei, in der wichtige Konstanten (»GEM betreffend) für C-Programmierer definiert sind (mit #define).

GEMDEFS.H

Hier gilt dasselbe wie unter »GEMBIND.H gesagte!

Gemdos

Abk. für -Graphic Environment Manager Disk Operating System- enthält die wichtigsten Funktionen zu Manipulationen von Dateien und Ein-/Ausgabe an »Tastatur, »Centronicsschnittstelle, »RS 232 und »Bildschirm. Gemdos wird über »Trap #1 aufgerufen und Rückgabewerte enthält immer das Register D0. In Assembler sieht der Aufruf folgendermaßen aus:

```
move.w #$20,-(sp)   * Zeichen auf den Stack schieben
move.w #$02,-(sp)   * Funktionsnummer von Cconout()
trap   #1           * Gemdos aufrufen
addq.l #$04,sp      * Stackpointer reparieren
```

Der Trap-Handler sichert nicht die Register D0-D2 und A0-A2.

Kurzübersicht der GEMDOS Funktionen mit Parametern:

Funktionsnummer	Funktion mit Parameter
$00 = 0	Pterm0()
$01 = 1	Cconin()
$02 = 2	Cconout(char)
$03 = 3	Cauxin()
$04 = 4	Cauxout(char)
$05 = 5	Cprnout(char)
$06 = 6	Crawio(wrd)
$07 = 7	Crawcin()
$08 = 8	Cnecin()
$09 = 9	Cconws(str)
$0A = 10	Cconrs(buf)
$0B = 11	Cconis()
$0E = 14	Dsetdrv(drv)
$10 = 16	Cconos()

Funktionsnummer	Funktion mit Parameter
$11 = 17	Cprnos()
$12 = 18	Cauxis()
$13 = 19	Cauxos()
$19 = 25	Dgetdrv()
$1A = 26	Fsetdta(ptr)
$20 = 32	Super(stack)
$2A = 42	Tgetdate()
$2B = 43	Tsetdate(date)
$2C = 44	Tgettime()
$2D = 45	Tsettime(time)
$2F = 47	Fgetdta()
$30 = 48	Sversion()
$31 = 49	Ptermres(keep,ret)
$36 = 54	Dfree(buf,drv)
$39 = 57	Dcreate(path)
$3A = 58	Ddelete(path)
$3B = 59	Dsetpath(path)
$3C = 60	Fcreate(name,attr)
$3D = 61	Fopen(name,handle)
$3E = 62	Fclose(handle)
$3F = 63	Fread(handle,count,buf)
$40 = 64	Fwrite(handle,count,buf)
$41 = 65	Fdelete(name)
$42 = 66	Fseek(offset,handle,mode)
$43 = 67	Fattrib(path,mode,mode)
$45 = 69	Fdup(stdhandle)
$46 = 70	Fforce(stdhandle,nonstdhandle)
$47 = 71	Dgetpath(pathbuf,drv)
$48 = 72	Malloc(amount)
$49 = 73	Mfree(addr)
$4A = 74	Mshrink(zero,mem,size)
$4B = 75	Pexec(mode,path,commandline,enviroment)
$4C = 76	Pterm(code)
$4E = 78	Fsfirst(spec,attr)
$4F = 79	Fsnext()
$56 = 86	Frename(zero,old,new)
$57 = 87	Fdatime(handle,buf,set)

Bei Gemdosaufrufen muß beachtet werden, daß sich die Ein-/Ausgaberoutinen für »Bildschirm, »RS 232, »Centronicsschnittstelle, »Tastatur durch folgende Tastendrücke beeinflussen lassen:

'Control S' hält die Ausgabe an.
'Control Q' setzt die Ausgabe fort.
'Control C' bricht die Ein-/Ausgabe ab und führt zum Programmabbruch.

Warnung:
Sie sollten beachten, nicht Langwörter zu testen, wenn ein Gemdosaufruf einen Fehler liefert, da die oberen 16 Bits mit irgendwelchen Werten gefüllt sein können. Testen sie nur Wörter.

Gemdos hat folgende »Handle vordefiniert:

Handle	Bedeutung
0	Standard Eingabegerät (Tastatur)
1	Standard Ausgabegerät (Bildschirm)
2	Standard Fehlerein-/ausgabegerät
3	RS 232 Ein-/Ausgabe
4	Standard 'LIST'-ausgabegerät (Drucker)

Diese Handle brauchen nicht mit *Fopen()* geöffnet und mit *Fclose()* geschlossen werden. Sie können sie mit *Fwrite()* und *Fread()* ansprechen.

$00 = 0 PtermO()

Mit dieser Funktion können Sie ein Programm verlassen, und in das aufrufende Programm zurückkehren. Der Speicher, der von ihren Programm belegt wurde, wird wieder frei gegeben.

Eingabeparameter: keine

Ausgabeparameter: keine

$01 = 1 zeichen = Cconin()
long zeichen;

Diese Funktion holt ein Zeichen aus dem Tastaturpuffer. Sie wartet solange bis ein Zeichen erhältlich ist. In den unteren 16 Bit wird das Zeichen zurückgegeben und in den oberen 16 Bit der »Scancode der Taste. Das empfangende Zeichen wird auf den Bildschirm ausgegeben. Bei Drücken der Taste Control C wird das Programm abgebrochen.

Eingabeparameter: keine

Ausgabeparameter:

long zeichen enthält Zeichen und Scancode

$02 = 2 **Cconout(chr)**
word chr;

Diese Funktion gibt ein Zeichen auf den Bildschirm aus und reagiert auf Control S, Control Q und Control C.

Eingabeparameter:

word chr auszugebendes Zeichen

Ausgabeparameter: keine

$03 = 3 **zeichen = Cauxin()**
long zeichen;

Liest ein Zeichen von der »RS 232 ein. Die Funktion kehrt zurück, wenn ein Zeichen empfangen wurde.

Eingabeparameter: keine

Ausgabeparameter:

word zeichen eingelesenes Zeichen

$04 = 4 **Cauxout(chr)**
word chr;

Gibt ein Zeichen auf der »RS 232 aus.

Eingabeparameter:

word chr auzugebendes Zeichen

Ausgabeparameter: keine

$05 = 05 **Cprnout(chr)**
word chr;

Gibt ein Zeichen an den Drucker aus.

Eingabeparameter:

word chr auszugebendes Zeichen

Ausgabeparameter: keine

$06 = 6 **Crawio(wrd)**
word 0 wrd;

Wenn 'wrd' ungleich 255 ist, wird 'wrd' auf den Bildschirm ausgegeben. Wenn 'wrd' gleich 255 ist, wird ein Zeichen von der Tastatur eingelesen.

Eingabeparameter:

word wrd zu sendendes Zeichen, wenn <> 255

Ausgabeparameter:

long zeichen wenn 'wrd' = 255, dann wird Zeichen und »Scancode zurückgegeben.

$07 = 7 **zeichen = Crawcin()**
long zeichen;

Liest ein Zeichen von der »Tastatur. In den oberen 16 Bit befindet sich der »Scancode.

Eingabeparameter: keine

Ausgabeparameter:

long zeichen enthält Zeichen und »Scancode

$08 = 8 **zeichen = Cnecin()**
long zeichen;

Hat den gleichen Effekt wie die Funktion *Crawcin()*. Sie hat nur bei MS-DOS Rechnern einen Sinn, wobei Tasten interpretiert werden, die der Atari ST nicht hat.

Eingabeparameter: keine

Ausgabeparameter:

long zeichen enthält ein Zeichen und »Scancode

$09 = 9 **Cconws(ptr)**
long ptr;

'ptr' zeigt auf einen String, der mit einer NULL endet. Von der Funktion wird der String auf den Bildschirm ausgeben. Sie reagiert auf Control C mit einem Programmabbruch.

Eingabeparameter:

long ptr Zeiger auf auzugebenen String

Ausgabeparameter: keine

$0A = 10 **Cconrs(buf)**
long buf;

Diese Funktion liest einen String von Tastatur ein. 'buf' zeigt auf einen Puffer, wobei das erste Byte angibt, wieviele Zeichen eingeben werden dürfen, und das zweite Byte, wieviel Zeichen eingegeben wurden. Ab dem dritten Byte werden eingegeben Zeichen deponiert. Die Funktion hört mit der Eingabe auf, wenn sie ein Byte weniger im Puffer hat, als im Byte NULL angeben wurde.

Folgende Control's werden von der Funktion interpretiert:

Control	Bedeutung
C	Programmabbruch
H	reagiert genauso wie Backspace
I	geht auf den nächsten Tabulator (acht Zeichen)
J	gleiche Reaktion wie bei Return (Eingabebeendigung)
R	eingegebene Zeile wird in der nächsten Zeile ausgegeben
U	Eingabe wird nicht beachtet und wird in der nächsten Zeile fortgesetzt
X	Eingabe löschen

Eingabeparameter:

long buf Puffer für einzugebende Zeichen

Ausgabeparameter: keine

$0B = 11 **status = Cconis()**
word status;

Diese Funktion stellt fest, ob ein Zeichen im Tastaturpuffer ist. Wenn der Rückgabewert ungleich Null ist, ist ein Zeichen im Tastaturpuffer. Wenn Null zurückgeliefert wird, ist kein Zeichen im Tastaturpuffer.

Eingabeparameter: keine

Ausgabeparameter:

word status Status der Konsole

$0E = 14 **olddrv = Dsetdrv(drv)**
word olddrv, drv;

Mit dieser Funktion können Sie das aktuelle Laufwerk einstellen. Für Laufwerk A: wird eine Null, für Laufwerk B: eine Eins übergeben usw. Die Funktion liefert zurück, welches Laufwerk vorher aktiv war.

Eingabeparameter:

word drv Laufwerksnummer

Ausgabeparameter:

word olddrv alte Laufwerksnummer

$10 = 16 **status = Cconos()**
word status;

Gibt an, ob ein Zeichen auf dem Bildschirm dargestellt werden kann. Bei Rückgabe von Null kann man kein Zeichen darstellen und bei der Rückgabe ungleich Null kann man ein Zeichen auf dem Bildschirm darstellen. Control C führt zu einem Programmabbruch.

Eingabeparameter: keine

Ausgabeparameter:

word status Status des Bildschirms

$11 = 17 **status = Cprnos()**
word status;

Mit dieser Funktion können Sie feststellen, ob ein Zeichen an den Drucker ausgegeben werden kann. Null bedeutet, der Drucker ist Busy, OFF-Line oder nicht eingeschaltet. Wenn ein Wert ungleich Null zurückgegeben wird, ist der Drucker empfangsbereit.

Eingabeparameter: keine

Ausgabeparameter:

word status Status der »Centronicsschnittstelle

$12 = 18 **status = Cauxis()**
word status;

Mit dieser Funktion können Sie feststellen, ob ein Zeichen aus dem Puffer der »RS 232 erhältlich ist. Null signalisiert, daß kein Zeichen erhältlich ist, und ungleich Null signalisiert, daß Zeichen erhältlich sind.

Eingabeparameter: keine

Ausgabeparameter:

word status Eingabestatus der »RS 232

$13 = 19 **status = Cauxout(chr)**
word status;

Mit dieser Funktion können Sie feststellen, ob die »RS 232 sendebereit ist. Null bedeutet, daß die »RS 232 nicht sendebereit ist Wenn ein Wert ungleich Null zurückgegeben wird, ist sie sendebereit.

Eingabeparameter: keine

Ausgabeparameter:

word status Ausgabestatus der »RS 233

$19 = 25 **drv = Dgetdrv()**
word drv;

Mit dieser Funktion können Sie feststellen, welches das aktuelle Diskettenlaufwerk ist. Die Laufwerksnummern entsprechen der Funktion *Dsetdrv()*.

Eingabeparameter: keine

Ausgabeparameter:

word drv Laufwerksnummer

$1A = 26 **Fsetdta(dta)**
long dta;

'dta' (Disk Transfer Address) zeigt auf einen 44 Byte langen Puffer, der auf einer geraden oder ungeraden Adresse beginnen kann. *Fsfirst()* und *Fsnext()* hinterlassen in diesem Puffer die Filesuchinformationen. Wenn Sie keinen Puffer eingestellt haben, be-

nutzt Gemdos einen default Puffer, der einen Offset von 80 Byte von der »Base Page hat. Mit *Fgetdta()* können Sie die Adresse des Puffers erfahren.

Eingabeparameter:

long dta Zeiger auf Disk Transfer Address

Ausgabeparameter: keine

$20 = 32 oldstack = Super(stack)
long oldstack, stack;

Sie können mit dieser Funktion feststellen, ob Sie im »User-Mode oder »Supervisor-Mode sind, und Sie können zwischen »User-Mode und »Supervisor-Mode hin- und herschalten.

Parameter	Bedeutung
1	wenn eine 1 zurückgegeben wird, sind Sie im »Supervisor-Mode und bei einer Null sind Sie im »User-Mode
0	Hiermit schalten Sie in den »Supervisor-Mode, und es wird der Userstackpointer zurückgegeben
»USP	wenn Sie den Userstackpointer übergeben, wird wieder in den User-Mode zurückgeschaltet und der Supervisorstackpointer zurückgegeben

Eingabeparameter:

long stack Parameter von *Super()*

Ausgabeparameter:

long oldstack Rückgabeparameter

$2A = 42 date = Tgetdate()
word date;

Mit dieser Funktion können Sie das aktuelle Datum erfahren.

Bits	Bedeutung
0 - 4	enthalten den Tag
5 - 8	enthalten den Monat
9 - 15	enthalten das Jahr. Es muß noch 1980 dazu addiert werden

Eingabeparameter: keine

Ausgabeparameter:

word date enthält das Datum

$2B = 43 Tsetdate(date)
word date;

Mit dieser Funktion können Sie das Datum einstellen. Die Bits von 'date' haben folgende Bedeutung.

Bits	Bedeutung
0 - 4	enthalten den Tag
5 - 8	enthalten den Monat
9 -15	enthalten das Jahr. Es muß noch 1980 abgezogen werden

Mit dieser Funktion beeinflussen Sie nicht das Datum, das Sie mit der Funktion *Getttime()* des »Extended Bios ergibt auslesen Benutzen Sie dazu die Funktion *Settime()* vom »Extended Bios. Die Funktion bemerkt nicht Fehler wie z.B.: 30.02.1986, sondern nur Fehler wie 32.02.1986. Dann wird -1 zurückgegeben.

Eingabeparameter:

word date das einzustellende Datum

Ausgabeparameter: keine

$2C = 44 time = Tgetime()
word time;

Mit dieser Funktion erhalten Sie das aktuelle Datum. Folgende Bedeutung haben die Bits:

Bit	Bedeutung
0 - 4	sind die Sekunden. Der Wert muß mit zwei multipliziert werden. Dadurch erhält man nur eine Genauigkeit von 2 Sekunden
5 - 10	enthält die Minuten
11 - 15	enthält die Stunden

Eingabeparameter: keine

Ausgabeparameter:

word time aktuelle Uhrzeit

$2D = 45 **Tsettime(time)**
word time;

Mit dieser Funktion können Sie die Uhrzeit einstellen. Es gilt das gleiche für das »Extended Bios wie bei der Funktion *Tsetdate()*. Wenn das Betriebssystem die Uhrzeit nicht akzeptiert hat, erhalten Sie eine -1 zurück. Die Bits von 'time' haben folgende Bedeutung:

Bit	Bedeutung
0 - 4	sind die Sekunden. Der Wert muß durch zwei dividiert werden. Dadurch erhält man nur eine Genauigkeit von 2 Sekunden
5 - 10	enthält die Minuten
11 - 15	enthält die Stunden

Eingabeparameter:

word time neue Zeit

Ausgabeparameter: keine

$2F = 47 **dta = Fgetdta()**
word dta;

Mit dieser Funktion können Sie feststellen, wo die »DTA (Disktransferaddress) des Betriebssystems ist. Siehe auch *Fsfirst()*

und *Fsetdta()*.

Eingabeparameter: keine

Ausgabeparameter:

long dta Adresse der »DTA

$30 = 48 **version = Sversion()**
word version;

Mit dieser Funktion können Sie die Versionsnummer des Betriebssystems erhalten. Die ROM-Version würde z.B.: $1300 ergeben.

Eingabeparameter: keine

Ausgabeparameter:

word version Versionsnummer des Betriebssystems

$31 = 49 **Ptermres(n_bytes, rc)**
long n_bytes;
word rc;

Mit dieser Funktion können Sie Programme beenden. 'n_bytes' gibt an, wieviele Bytes ab Anfang des Programmes inklusive »Basepage reserviert werden sollen. Dies hat einen besonderen Sinn, so können Sie z.B. eigene Interruptroutinen ins Betriebssystem einbinden, ohne daß Sie beim Beenden des Programmes gelöscht werden, wie mit der Funktion *Pterm0()* der Fall ist. Mit 'rc' können Sie dem aufrufenden Programm einen Wert zurückgeben.

Eingabeparameter:

long n_bytes Länge des zu schützenden Programmes
word rc Rückgabewert an das aufrufende Programm

Ausgabeparameter: keine

```
$36 = 54        error = Dfree(pbuffer, drv)
long                            pbuffer;
word            error,                  drv;
```

Diese Funktion sagt Ihnen, wieviel Speicher auf der Diskette frei ist. 'pbuffer' ist ein Zeiger, der auf vier Langwörter zeigt:

Offset	Bedeutung
$00 = 0	Menge der freien Bytes auf Diskette
$04 = 4	Menge der freien »Cluster
$08 = 8	Bytes per »Sektor
$0c = 12	Menge der Sektoren pro Cluster

Mit 'drv' geben Sie die Diskettenstation, Harddisk usw. an.

0 = aktuelles Laufwerk, 1 = Laufwerk A:, 2 = Laufwerk B: usw.

'error' kann sein:

error	Bedeutung
-46	unbekannte Laufwerksnummer

Eingabeparameter:

long pbuffer	zeigt auf 16 freie Bytes bzw. 4 Langwörter
word drv	gibt an, auf welchem Laufwerk *Dfree()* ausgeführt werden soll

Ausgabeparameter:

word error	gibt an, ob ein Fehler aufgetreten ist

```
$39 = 57        error = Dcreate(path)
word            error;
long                            path;
```

Diese Funktion erlaubt es einen Ordner zu erzeugen. 'path' zeigt auf folgendes Format: **\Ordner**
Dabei ist zubeachten, daß Sie maximal 10 Ordner ineinander verschachtelt und 32 Ordner auf einer Diskette erzeugt werden dürfen.
Wenn Sie einen Ordner2 im Ordner1 erzeugen wollen, müßen Sie erst den obersten Ordner erzeugen, falls er noch nicht existiert. Das würde folgendermaßen aussehen:

```
Dcreate("\Ordner1");
Dcreate(" \Ordner1\Ordner2");
```

Es können folgende Fehler auftreten:

error	Bedeutung
-34	Pfad nicht gefunden
-36	Zugriff nicht möglich

Eingabeparameter:

long path zeigt auf den Suchpfad und Ordnernamen

Ausgabeparameter:

word error Fehler dieser Funktion

$3A = 58 error = Ddelete(path)
word error;
long path;

Mit dieser Funktion können Sie einen Ordner löschen. 'path' hat das gleiche Format wie bei *Dcreate()*.

'error' kann sein:

error	Bedeutung
-34	Pfad nicht gefunden
-36	Zugriff nicht möglich
-65	interner Fehler

Eingabeparameter:

long path zeigt auf den Suchpfad und den zulöschenden Ordner

Ausgabeparameter:

word error Fehler der Funktion

$3B = 59 error = Dsetpath(path)
word error;
long path;

Mit dieser Funktion können Sie das Directory bzw. den Suchpfad

einstellen. 'path' zeigt auf den Suchpfad. Wenn das aktuelle Directory z.B.: Ordner2 sein soll, muß 'path' auf folgenden String zeigen "\Ordner1\Ordner2".

Es kann folgender Fehler auftreten:

error	Bedeutung
-34	Pfad nicht gefunden

Eingabeparameter:

long path zeigt auf den Suchpfad

Ausgabeparameter:

word error Fehler der Funktion

```
$3C = 60        handle = Fcreate(name, attr)
long                             name;
word            handle,                 attr;
```

Mit dieser Funktion können Sie eine Datei erzeugen, die gleichzeitig eröffnet wird. Sie brauchen nicht mehr *Fopen()* aufzurufen Falls die Datei schon existieren sollte, wird Sie auf Null Bytes Länge gekürzt. Sie können auch mit *Fcreate()* der Diskette einen Namen geben. 'name' hat folgendes Format:
12345678.123
Es sind folgende Zeichen im Namen zugelassen:
0-9,A-Z,'_' und der Punkt, der markiert, daß danach eine bis zu drei Zeichen lange »Extension folgt. Um innerhalb eines Ordners eine Datei zu erzeugen, müssen Sie den Pfad vorher mit *Dsetpath()* einstellen.

Mit 'attr' können Sie folgendes einstellen:

attr	Bedeutung
$00 = 0	normale schreib/lese Datei
$01 = 1	nurlese Datei
$02 = 2	versteckte Datei
$04 = 4	versteckte Systemdatei
$08 = 8	Diskettenname

In 'handle' wird die »Handlenummer der erzeugten Datei zurückgegeben.

Fcreate() kann folgende Fehler zurückgeben:

Fehler	Bedeutung
-34	Pfad nicht gefunden
-35	keine Handle mehr übrig
-36	Zugriff nicht möglich

Eingabeparameter:

long name zeigt auf einen bis zu zwölf Byte langen String
word attr ist das Attribut der erzeugten Datei

Ausgabeparameter:

word handle »Handlenummer

$3D = 61 handle = Fopen(pname, mode)
word handle, mode;
long pname;

Mit *Fopen()* können Sie Dateien öffnen. Die Handle null bis vier sind vordefiniert (siehe dazu den Anfang der Gemdosbeschreibung). Sie können aber trotzdem noch Geräte öffnen:

pname	handle	Bedeutung
CON:	-1	Tastatur und Bildschirm
AUX:	-2	RS 232
PRN:	-3	Centronicsschnittstelle

'pname' zeigt auf den Namen, der so aufgebaut ist wie in *Create()* beschrieben. Allerdings können Sie auch das Laufwerk im Namen angeben: Fopen("A:\TEST",0)
Es können auch Ordnernamen bei 'pname' angegeben werden. Fopen("A:\ORDNER\TEST",0) funktioniert nicht. Es wird der Fehler -33 zurückgegeben. Wenn Sie innerhalb von Ordnern eine Datei öffnen wollen, müssen Sie vorher mit *Dsetpath()* den richtigen Pfad einstellen. Dann können Sie das Kommando so ausführen, wie oben beschrieben.

Mit 'mode' können Sie die Zugriffsart auf die Datei bestimmen:

mode	Bedeutung
0	nur lesen
1	nur schreiben
2	schreiben und lesen

Es können folgende Fehler auftreten:

Fehler	Bedeutung
-33	Datei nicht gefunden
-35	keine Handle übrig
-36	Zugriff nicht möglich

Eingabeparameter:

long pname zeigt auf den Namen der Datei
word mode Zugriffsmode

Ausgabeparameter:

word handle »Handlenummer

$3E = 62 error = Fclose(handle)
word error, handle;

Hiermit können Sie die Dateien schließen, die Sie mit *Fopen()* oder *Create()* geöffnet haben.

'error' kann sein:

error	Bedeutung
-37	falscher Handle

Eingabeparameter:

word handle Handlenummer

Ausgabeparameter:

word error Fehler der Funktion

$3F = 63 error = Fread(handle, count, buf)
word handle;
long error, count, buf;

Hiermit können Sie können von einer Datei lesen, die Sie mit *Fopen()* geöffnet haben.

'handle' ist die Handlenummer, die *Fopen()* zurückgegeben hat.

'count' sagt, wieviele Bytes gelesen werden sollen.

'buf' zeigt auf den Puffer, wohin die Daten gepackt werden sollen.

Bei erfolgreicher Operation gibt die Funktion einen Zeiger auf das nächste Byte der Datei zurück.

Es können folgende Fehler auftreten:

error	Bedeutung
-37	falscher Handle

Eingabeparameter:

```
word handle      Handlenummer
long count       Menge der Daten
long buf         Zeiger auf den Puffer der Daten
```

Ausgabeparameter:

```
word error       Fehler der Funktion
```

```
$40 = 64        error = Fwrite(handle, count, buf)
word                            handle;
long            error,                  count, buf;
```

Mit dieser Funktion ist es möglich, eine Datei zu lesen.

'handle' ist die Handlenummer, die *Fopen()* oder *Fcreate()* zurückgegeben haben.

'count' sagt, wieviele Bytes geschrieben werden sollen.

'buf' zeigt auf die Daten, die geschrieben werden sollen.

Bei erfolgreicher Operation gibt die Funktion einen Zeiger auf das nächste Byte in der Datei zurück.

'error' kann sein:

error	Bedeutung
-36	Zugriff nicht möglich
-37	falscher Handle

Eingabeparameter:

word handle Handlenummer

Ausgabeparameter:

word error Fehler der Funktion

$41 = 65 error = Fdelete(name)
long name;

Fdelete() ermöglicht es, eine Datei zu löschen. Sie müssen vorher, falls sich der Name in einem Ordner befindet, mit *Dsetpath()* den Pfad einstellen.

'name' zeigt auf den Namen der zu löschenden Datei. Der Name ist genauso aufgebaut wie bei *Fcreate()* beschrieben.

error	Bedeutung
-33	Datei nicht gefunden
-36	Zugriff nicht möglich

Eingabeparameter:

long name Name der zu löschenden Datei

Ausgabeparameter:

word error Fehler der Funktion

$42 = 66 error = Fseek(softs, handle, smode)
long error softs;
word handle, smode;

Diese Funktion erlaubt, sich relativ bzw. absolut in einer Datei zu bewegen.

'softs' gibt an, um wieviele Bytes sich der sich der Dateizeiger verändern soll.

'handle' ist die Handlenummer, die bei *Fcreate()* oder *Fopen()* zurückgegeben wurde.

Mit 'smode' geben Sie an, ob relativ oder absolut positioniert werden soll:

smode	Bedeutung
0	'softs' Bytes vom Begin der Datei (absolut)
1	verschiebt den Dateizeiger um 'softs' Bytes in Richtung Ende der Datei
2	verschiebt den Dateizeiger in Richtung des Anfang, wobei 'softs' nur negativ sein darf

Bei erfolgreicher Operation wird der Dateizeiger zurückgegeben.

Es können folgende Fehler auftreten:

error	Bedeutung
-32	falsche Funktion
-37	falscher Handle

Eingabeparameter:

long softs	Offset
word handle	Handlenummer
word smode	Modus von *Fseek()*

Ausgabeparameter:

word error	Fehler der Funktion

```
$43 = 67        attr = Fattrib(pname, wrt, mode)
long                          pname,
word            attr,                 wrt, mode;
```

Mit dieser Funktion können Sie die Attribute einer Datei einstellen. Das Attribut eines Diskettennamen und eines Ordners kann nicht verändert werden.

'pname' zeigt auf den Namen der Datei. Der Name ist genauso aufgebaut wie bei *Fopen()* beschrieben.

'wrt' gibt an, was die Funktion ausführen soll:

wrt	Bedeutung
0	Dateiattribut zurückgeben
1	Dateiattribut einstellen

'mode' gibt den Modus der Datei an:

mode	Bedeutung
$00 = 0	schreib/lese Datei
$01 = 1	nurlese Datei
$02 = 2	versteckte Datei
$04 = 4	versteckte Systemdatei
$08 = 8	Eintrag enthält den Diskettennamen
$10 = 16	Eintrag ist ein Ordner
$20 = 32	Datei, die im Multi-User-Betrieb geschlossen wurde (beim Atari nicht benutzt)

Es können folgende Fehler auftreten:

Fehler	Bedeutung
-33	Datei nicht gefunden
-34	Pfad nicht gefunden

Eingabeparameter:

long pname	Name der Datei
word wrt	Funktionsausführung
word mode	Modus von *Fattrib()*

Ausgabeparameter:

word attr	Fehler oder Dateiattribut

$45 = 69 **handle = Fdup(stdhnd)**
word handle, stdhnd;

Mit dieser Funktion können Sie einen Standard Ein-/Ausgabe Handle (wie am Anfang vom Gemdos beschrieben) über einen nicht Standard Handle ansprechen, der von *Fdup()* zurückgegeben wird.

'stdhnd' kann ein Handle von null bis vier sein.

Es können folgende Fehler auftreten:

Fehler	Bedeutung
-35	keine Handle übrig
-37	falscher Handle

Eingabeparameter:

word stdhnd Handlenummer der Standardgerätes

Ausgabeparameter:

word handle Handlenummer oder Fehler

$46 = 70 **error = Fforce(stdhnd, nsthnd)**
word error, stdhnd, nsthnd;

Fforce() erlaubt ein Standard Ein-/Ausgabe Handle (wie am Anfang vom Gemdos beschrieben) umzuleiten in eine Diskettendatei.

'stdhnd' kann ein Handle von null bis vier sein.

'nsthnd' ist ein Handle, der *Fopen()* oder *Create()* zurückgegeben wurde.

Es kann folgender Fehler auftreten:

error	Bedeutung
-37	falscher Handle

Eingabeparameter:

word stdhnd Handlenummer eines Standardgerätes
word nsthnd Handlenummer, auf die umgeleitet werden soll

Ausgabeparameter:

word error Fehler

$47 = 71 **error = Dgetpath(pathbuf, drv)**
word error, drv;
long pathbuf;

Mit dieser Funktion erhalten Sie den aktuellen Suchpfad, der z.B.: mit *Dsetpath()* gesetzt wurde.

'pathbuf' muß auf einen 64 Byte langen Puffer zeigen. In diesem Puffer wird der Suchpfad zurückgegeben.

'drv' gibt an, von welchem Laufwerk der Suchpfad zurückgegeben werden soll:

drv	Bedeutung
0	aktuelles Laufwerk
1	Laufwerk A:
2	Laufwerk B:
3	Laufwerk C:
.	.
.	.

'error' kann sein:

error	Bedeutung
-46	falsches Laufwerk

Eingabeparameter:

long pathbuf Zeiger auf einen 64 Byte langen Puffer
word drv Laufwerksnummer

Ausgabeparameter:

word error Fehler

$48 = 72 adr = Malloc(nbytes)
long adr, nbytes;

Malloc() erlaubt, sich vom Betriebssystem Speicher zu holen.

'nbytes' gibt an, wieviele Bytes Speicher angefordert werden sollen. Wenn Sie -1 übergeben, gibt die Funktion zurück, wieviel Speicher noch frei ist.

In 'adr' steht die Anfangsadresse des Speichers bzw. die Menge an freiem Speicher. Wenn der Aufruf mißglückt, wird eine Null zurückgegeben.

Eingabeparameter:

long nbytes Menge des benötigten Speichers bzw. -1

Ausgabeparameter:

long adr Adresse des Speichers, freier Speicher oder Null

```
$49 = 73        error = Mfree(adr)
word            error;
long                          adr;
```

Mit *Mfree()* können Sie den Speicher, den Sie mit *Malloc()* angefordert oder mit *Mshrink()* geschützt haben, wieder an das Betriebssystem zurückgeben.

'adr' enthält die Startadresse des Speichers.

'error' kann sein:

error	Bedeutung
-40	falsche Speicherblockadresse

Eingabeparameter:

long adr Startadresse des freizugebenden Speichers

Ausgabeparameter:

word error Fehler

```
$4A = 74        error = Mshrink(res, mp, size)
word            error,          res,
long                                  mp, size;
```

Mit *Mshrink()* können Sie einen von ihnen gewählten Speicherbereich schützen. Dies findet hauptsächlich Anwendung bei dem Start eines Programmes, weil »Gemdos den gesamten Speicherbereich dem Programm zu Verfügung stellt und somit kein Speicher mit *Malloc()* erhältlich ist. Die Programmlänge steht in der »Basepage (siehe dazu auch unter »Programmbegin nach).

'res' muß Null sein, hat aber keine besondere Bedeutung. Es kann sein, das *Mshrink()* in der C-Includedatei *osbind.h* folgendermaßen definiert ist: **Mshrink(mp, size)**

'mp' ist die Startadresse des Speicherbereich der geschützt werden soll.

'size' ist die Länge des zu schützenden Speicherbereiches.

'error' kann folgende Werte annehmen:

error	Bedeutung
-40	falsche Speicherblockadresse
-67	schwerer Fehler

Eingabeparameter:

word res	muß Null sein
long mp	Startadresse des zu schützenden Speichers
long size	Länge des zu schützenden Speichers

Ausgabeparameter:

word error	Fehler

```
$4B = 75        base = Pexec(load, pcspec, pcmdln, penvstr)
long            base,              pcspec, pcmdln, penvstr;
word                        load;
```

Diese Funktion erlaubt Ihnen, ein anderes Programm zu starten.

'load' kann sein:

load	Bedeutung
0	das Programm wird geladen und gestartet
3	Programm wird nur geladen. Es wird die Basepageadresse zurückgegeben

'pcspec' zeigt auf den Namen des zu startenden Programmes im Format von *Fopen()*.

'pcdln' zeigt auf die Kommandozeile.

'penvstr' zeigt auf das »Environment

Eingabeparameter:

word load	gibt an, wie das Programm geladen werden soll
long pcspec	zeigt auf den Namen des Programmes

long pcdln zeigt auf die Kommandozeile
long penvstr zeigt auf den Environmentstring

Ausgabeparameter:

word base bei 'load' = 0 ist 'base' gleich Null und bei 'load' = 3 zeigt 'base' auf die »Basepage. -1 ist 'base', wenn ein Fehler aufgetreten ist.

$4C = 76 **Pterm(code)**
word code;

Mit *Pterm()* wird ein Programm wie mit *Pterm0()* verlassen, mit dem Unterschied, daß Sie einen Parameter ihrer Wahl zurückgeben können. Der Wert wird im Register D0 an das aufrufende Programm zurückgegeben.

'code' ist der von ihnen gewählte Rückgabeparameter.

Eingabeparameter:

word code Rückgabeparameter

Ausgabeparameter: keine

$4E = 78 **error = Fsfirst(name, attr)**
long name;
word error, attr;

Diese Funktion erlaubt es, eine Datei im Directory zu suchen. Weiterhin wird die »DTA mit Daten aufgefüllt:

Offset	Bedeutung
0 - 20	reserviert für Gemdos
22 - 23	Zeit der Speicherung der Datei. Das Format ist wie bei *Tgettime()*
24 - 25	Datum der Speicherung der Datei. Das Format ist das gleiche wie bei *Tgetdate()*
26 - 29	enthält die Länge der Datei
30 - 43	enthält den Namen der Datei

'name' gibt die gesuchte Datei an, wobei Sie mit »Wildcards arbeiten können. Wenn Sie z.B.: '*.*' angeben, würde die erste Datei mit dem entsprechenden Attribut, wie in 'attr' beschrieben, gefunden werden. Mit *Fsnext()* können Sie die nächste Datei suchen, auf die der Name paßt.

Mit 'attr' können Sie folgendes einstellen:

attr	Bedeutung
$00 = 0	normale schreib/lese Datei
$01 = 1	nurlese Datei
$02 = 2	versteckte Datei
$04 = 4	versteckte Systemdatei
$08 = 8	Diskettenname
$10 =16	Ordner
$20 =32	Dateien, die unter einem Multi-User-Betrieb geschlossen wurden (beim Atari nicht verwendet)

Es können folgende Fehler auftreten:

error	Bedeutung
-33	Datei nicht gefunden
-49	keine weiteren Dateien

Eingabeparameter:

long name	Name der gesuchten Datei
word attr	gibt an, nach welcher Art von Datei gesucht wird

Ausgabeparameter:

word error	Fehler

$4F = 79 **error = Fsnext()**
word error;

Fsnext() sucht nach der nächsten Datei, auf die die Parameter, die Sie in *Fsfirst()* angegeben haben passen. Dabei wird die DTA mit den Daten der gefundenen Datei aufgefüllt.

'error' kann sein:

error	Bedeutung
-49	keine weiteren Dateien

Eingabeparameter: keine

Ausgabeparameter:

word error Fehler

```
$56 = 86        error = Frename(res, oldname, newname)
word            error,          res;
long                                oldname, newname;
```

Frename() erlaubt das Umbenennen einer Datei. Die beiden Namen bauen sich auf wie bei *Fopen()* beschrieben.

'res' muß Null sein.

'oldname' zeigt auf den Namen der umzubenennenden Datei.

'newname' zeigt auf den neuen Namen.

'error' kann sein:

error	Bedeutung
-34	Pfad nicht gefunden
-36	wird zurückgegeben, wenn die Datei den nurlese Status hat

Eingabeparameter:

word res ist immer eine Null
long oldname zeigt auf den alten Namen
long newname zeigt auf den neuen Namen

Ausgabeparameter:

word error Fehler

```
$57 = 87        Fdatime(handle, buf, set))
word                    handle,      set;
long                            buf;
```

Mit dieser Funktion können Sie Datum und Uhrzeit des letzten Schreibzugriffs auf einer Datei ermitteln, bzw. ein neues Datum

und eine neue Uhrzeit einstellen. Bei der Definition dieser Funktion wurden aus Kompatibilätsgründen zum GEM auf MSDOS Rechnern in der C-Includedatei *osbind.h* zwei Parameter vertauscht. Es sieht dort folgendermaßen aus: **Fdatime(buf, handle, set)**

'handle' wird durch *Create()* oder *Fopen()* zurückgegeben und bezeichnet die Datei, mit der bearbeitet werden soll.

'buf' zeigt auf einen Puffer, in dem das erste Wort die Zeit und das zweite Wort das Datum ist.

'set' kann sein:

set	Bedeutung
0	Zeit und Datum sollen neu eingestellt werden
1	Zeit und Datum wird im Puffer abgelegt

Eingabeparameter:

word handle	Handle-Nummer
long buf	zeigt auf einen vier Byte langen Puffer
word set	gibt an, was *Fdatime()* ausführen soll

Ausgabeparameter: keine

Generalized Drawing Primitive

engl., Bezeichnung für eine Reihe von Grundroutinen des »VDI zum Zeichnen verschiedener grafischer Grundelemente. Dazu zählen:

- Balken	siehe »*v_bar()*
- Kreis	siehe »*v_circle()*
- Kreisausschnitt	siehe »*v_arc()*
- Tortenstück	siehe »*v_pieslice()*
- Ellipse	siehe »*v_ellipse()*
- Ellipsenausschnitt	siehe »*v_ellarc()*
- elliptisches Tortenstück	siehe »*v_ellpie()*
- Rechteck mit abgerundeten Ecken	siehe »*v_rbox()*
- Rechteck mit abgerundeten Ecken und Füllmuster	siehe »*v_rfbox()*
- Grafiktext mit Randausgleich	siehe »*v_justified()*

Sehen Sie bitte unter den einzelnen Stichwörtern nach!

Graphics Environment Manager

Siehe »GEM

Graphics Library

Diese Bibliothek enthält ein Sammlung von Routinen, die verschiedensten Zwecken dienen. Ihre Bestandteile sind folgende:

-»*graf_dragbox()*	Bewegt mit der Maus ein Rechteck, wobei der Mauszeiger immer an derselben Position im Rechteck bleibt
-»*graf_growbox()*	Zeichnet ein sich ausdehnendes Rechteck auf den »Bildschirm
-»*graf_handle()*	Gibt die Nummer der »virtuellen Workstation zurück, die von der Graphics Library (und »AES) benutzt wird
-»*graf_mkstate()*	Gibt die Position des Mauszeigers, den Status der Maustasten und der Tastatur zurück
-»*graf_mouse()*	Zeigt oder versteckt den Mauszeiger und legt dessen Form fest
-»*graf_movebox()*	Zeichnet ein sich bewegendes Rechteck mit konstanter Größe
-»*graf_rubberbox()*	Zeichnet ein Rechteck mit fester Ecke links oben und einer (mit der »Maus) verschiebbaren Ecke rechts unten
-»*graf_shrinkbox()*	Zeichnet ein sich verkleinerndes Rechteck
-»*graf_slidebox*()	Verschiebt ein Rechteck innerhalb eines bestimmten Bereichs synchron zur Mausbewegung
-»*graf_watchbox()*	Verfolgt den Mauszeiger und verändert den Status eines »Objekts, wenn der Mauszeiger es berührt oder verläßt

Um mit den Erläuterungen zu den einzelnen Routinen etwas anfangen zu können, sollte man mit der Bedeutung folgender Stichworte vertraut sein:

- »GEM
- »AES
- »VDI
- »handle
- »Worstation
- »virtuell Workstation
- »Maus
- »Objekt

Graf_dragbox

Zur »Graphics Library gehörende Routine

```
$47 = 71        gr_dreturn = graf_dragbox(gr_dwidth, gr_dheight,
word            gr_dreturn,               gr_dwidth, gr_dheight;
                                          gr_dstartx, gr_dstarty,
word                                      gr_dstartx, gr_dstarty;
                                          gr_dboundx, gr_dboundy,
word                                      gr_dboundx, gr_dboundy;
                                          gr_dboundw, gr_dboundh,
word                                      gr_dboundw, gr_dboundh;
                                          &gr_dfinishx, &gr_dfinishy)
word *                                     gr_dfinishx,  gr_dfinish;
```

Mit dieser Routine ist es möglich, den Benutzer ein Rechteck auf dem »Bildschirm bewegen zu lassen. Solange der Anwender den Mausknopf gedrückt läßt, kann er das Rechteck mit der »Maus innerhalb vorgegebener Grenzen bewegen. Mit Loslassen des Mausknopfs kehrt die Kontrolle zur aufrufenden Routine zurück. Diese Routine wird beispielsweise von »AES benutzt, wenn der Benutzer ein Fenster bewegt. Der begrenzende Rahmen ist dabei der gesamte »Bildschirm. Die Position des Mauszeigers innerhalb des Rechtecks verändert sich nicht.

Eingabeparameter:

word gr_dwidth	Die Breite des zu verschiebenden Rechtecks (in »Pixeln)
word gr_dheight	Die Höhe des Rechtecks (in »Pixeln)
word gr_dstartx	Die X-Koordinate des Rechtecks zu Beginn der Aktion
word gr_dstarty	Die Y-Koordinate des Rechtecks zu Beginn der Aktion
word gr_dboundx	Die X-Koordinate des begrenzenden Rechtecks
word gr_dboundy	Die Y-Koordinate des begrenzenden Rechtecks
word gr_dboundw	Die Breite des begrenzenden Rechtecks
word gr_dboundh	Die Höhe des begrenzenden Rechtecks
word *dfinishx word *dfinishy	Adressen der Rückgabewerte (s. u.)

```
contrl(0)   =  71
contrl(1)   =   8
contrl(3)   =   0

int_in(0)   =  gr_dwidth
int_in(1)   =  gr_dheight
int_in(2)   =  gr_dstartx
int_in(3)   =  gr_dstarty
int_in(4)   =  gr_dboundx
int_in(5)   =  gr_dboundy
int_in(6)   =  gr_dboundw
int_in(5)   =  gr_dboundh
```

Ausgabeparameter:

```
word gr_dreturn     0 = Ein Fehler ist passiert, n = alles Ok

word gr_dfinishx    Die X-Koordinate der Maus, wenn der Benutzer
                    den Mausknopf losläßt

word gr_dfinishy    Die Y-Koordinate der Maus, wenn der Benutzer
                    den Mausknopf losläßt

contrl(2)   =   3
contrl(4)   =   0

int_out(0)  =  gr_dreturn
int_out(1)  =  gr_dfinishx
int_out(2)  =  gr_dfinishy
```

Graf_growbox

Zur »Graphics Library gehörende Routine

```
$49 = 73        gr_greturn = graf_growbox(gr_gstx, gr_gsty, gr_gstwidth,
word            gr_greturn,               gr_gstx, gr_gsty, gr_gstwidth;
                                          gr_gstheight, gr_finix,
word                                      gr_gstheight, gr_finix;
                                          gr_gfiny, gr_gfinwidth,
word                                      gr_gfiny, gr_gfinwidth;
                                          gr_gfinheight)
word                                      gr_gfinheight;
```

Außer hübschem Aussehen hat diese Routine wenig Gebrauchsnutzen: sie zeichnet ein sich stetig vergrößerndes Rechteck. Meistens soll diese Routine "Etwas sich Öffnendes" symbolisieren. Da hierbei oft nur Zeit verschwendet wird, sollte man sich überlegen, ob das Sinn macht.

Eingabeparameter:

word gr_gstx	Die X-Koordinate zu Beginn für das erste gezeichnete Rechteck
word gr_gsty	Die Y-Koordinate zu Beginn für das erste gezeichnete Rechteck
word gr_gstwidth	Die Breite zu Beginn für das erste gezeichnete Rechteck
word gr_gstheight	Die Höhe zu Beginn für das erste gezeichnete Rechteck
word gr_finx	Die X-Koordinate des Rechtecks am Schluß
word gr_finy	Die Y-Koordinate des Rechtecks am Schluß
word gr_finwidth	Die Breite des Rechtecks am Schluß
word gr_finheight	Die Höhe des Rechtecks am Schluß

```
contrl(0)   =  73
contrl(1)   =   8
contrl(3)   =   0

int_in(0)   =  gr_gstx
int_in(1)   =  gr_gsty
int_in(2)   =  gr_gstwidth
int_in(3)   =  gr_gstheight
int_in(4)   =  gr_gfinx
int_in(5)   =  gr_gfiny
int_in(6)   =  gr_gfinwidth
int_in(7)   =  gr_gfinheight
```

Ausgabeparameter:

word gr_greturn 0 = Ein Fehler ist vorgekommen, n = alles Ok

```
contrl(2)   =   1
contrl(4)   =   0

int_out(0)  =  gr_greturn
```

Graf_handle

Zur »Graphics Library gehörende Routine

```
$4D = 77        gr_handle = graf_handle(&gr_hwchar, &gr_hhchar,
word            gr_handle;
word *                                  gr_hwchar,  gr_hhchar;
                                       &gr_hwbox, &gr_hhbox)
word *                                  gr_hwbox,  gr_hhbox;
```

Mit dieser Funktion bekommt man den »handle der für »GEM VDI geöffneten »Workstation zurück (das ist in diesem Fall der »Bildschirm). Diese »virtual Workstation steht auch dann zur Verfügung, wenn noch keine eigene geöffnet wurde. »VDI und »AES und alle GEM-Programme benutzen diesen »handle, wann immer sie auf den »Bildschirm zugreifen. Ferner soll diese Funktion auch verschiedene Werte für die Buchstabengröße zurückgeben (siehe »Font).

Allerdings sind die Werte, die man für den Zeichensatz bekommt, falsch. So wird beispielsweise die Breite der Zeichen in »Drop-Down-Menüs und »Dialogboxen mit eins angegeben. Die richtigen Werte bekommen Sie mit der Funktion »*vqt_attributes()*.

Trotzdem ist die Routine wichtig: denn um eine »virtual Workstation öffnen zu können, muß bereits eine geöffnet sein. Den »handle dieser bereits geöffneten Arbeitsstation bekommen Sie mit *graf_handle()*. Siehe auch »*v_opnvwk()*.

Eingabeparameter:

```
word *gr_hwchar
word *gr_hhchar
word *gr_hwbox
word *gr_hhbox      Zeiger auf Rückgabewerte (s.u.)

contrl(0)   =  77
contrl(1)   =   0
contrl(3)   =   0
```

Ausgabeparameter:

```
word gr_handle      Der »GEM VDI Handle (siehe unter »Handle)

                    Die folgenden vier Parameter  werden vom Be-
                    triebssystem nicht korrekt zurückgegeben:
word gr_hwchar      Die Breite der Zeichen im System-Zeichensatz
                    (siehe »Font) in Pixeln

word gr_hhchar      Die Höhe der Zeichen im System-Zeichensatz
```

word gr_hhwbox Die Breite des Rechtecks, das ein Zeichen des System-Zeichensatzes aufnehmen kann (d. h. das Zeichen selbst plus einem Zwischenraum) in Pixeln

word gr_hhbox Die Höhe des Rechtecks (s. o.)

```
contrl(2)   =   5
contrl(4)   =   0

int_out(0)  =  gr_handle
int_out(1)  =  gr_hwchar
int_out(2)  =  gr_hhchar
int_out(3)  =  gr_hwbox
int_out(4)  =  gr_hhbox
```

Graf_mkstate

Zur »Graphics Library gehörende Routine

```
$4F = 79        gr_mresvd = graf_mkstate(&gr_mkmx, &gr_mkmy,
long                                      gr_mkmx,  gr_mkmy;
                                         &gr_mkmstate, &gr_mkkstate)
word *                                    gr_mkmstate,  gr_mkkstate;
```

Mit dieser Routine kann die Position des Mauszeigers, der Zustand der Maustasten sowie der Zustand der Tastatur (Shift, Control etc.) abgefragt werden.

Eingabeparameter:

word *gr_mkmx
word *gr_mkmy
word *gr_mkmstate
word *gr_mkkstate Adressen der Variablen für Rückgabewerte (s. u.)

```
contrl(0)   =  79
contrl(1)   =   0
contrl(3)   =   0
```

Ausgabeparameter:

word gr_mkresvd Sinnloser Wert, immer gleich eins

word gr_mkmx Die X-Koordinate des Mauszeigers

word gr_mkmy Die Y-Koordinate des Mauszeigers

word gr_mkmstate — Der aktuelle Zustand der Mausknöpfe - was dies angeht, so gilt für **alle** Routinen, die den entsprechenden Wert zurückgeben, folgende Zuordnung:

Bit	Mausknopf
0	linker Mausknopf
1	zweiter Knopf von Links
2	dritter Knopf von Links usw. (bis 16!)

Ist das Bit gesetzt, so ist die entsprechende Taste gedrückt, wenn nicht, ist die Taste nicht gedrückt (zum Zeitpunkt der Abfrage).

word gr_mkkstate — Der Zustand der Tastatur - wenn das Bit gesetzt ist, so wurde die entsprechende Taste gedrückt (Capslock fehlt bei dieser Funktion!):

Bit	Bedeutung
0	Rechte Shifttaste
1	Linke Shifttaste
2	Controltaste
3	Alternatetaste

```
contrl(2)   =   5
contrl(4)   =   0

int_out(0)  =  gr_mkresvd
int_out(1)  =  gr_mkmx
int_out(2)  =  gr_mkmy
int_out(3)  =  gr_mkmstate
int_out(4)  =  gr_mkkstate
```

Graf_mouse

Zur »Graphics Library gehörende Routine

```
$4E = 78        gr_moreturn = graf_mouse(gr_monumber, gr_mofaddr)
word            gr_moreturn,             gr_monumber;
word *                                                gr_mofaddr;
```

Mit dieser Funktion ist es möglich, die Form des »Mauszeigers zu verändern. Da gibt es zum einen acht vordefinierte Formen (s. u.), aber es ist auch möglich, eigene Figuren zu erzeugen (wie die Daten organisiert sein müssen, steht unter »Mauszeiger).

Ferner kann man den »Mauszeiger auch ein- und ausschalten. Dies ist dann nötig, wenn man auf dem »Bildschirm zeichnet oder printet. Denn immer wenn man den »Mauszeiger bewegt, wird der Hinter-

grund des Zeigers zwischengespeichert. Bei der nächsten Bewegung wird der Hintergrund wieder zurück auf den »Bildschirm gepackt. Wurde nun über den Maussprite gezeichnet, so wird bei der nächsten Bewegung der alte Hintergrund inkorrekterweise restauriert. Also muß bei jeder Veränderung auf dem Bildschirm der »Mauszeiger "versteckt" werden. Dabei sollte streng darauf geachtet werden, daß sich die Befehle zum Zeigen und Verstecken des Mauscursors die Waage halten, denn wenn man den Befehl zum Verstecken fünfmal gibt, so muß der Befehl zum Zeigen ebenfalls fünfmal erfolgen, damit man den Cursor wieder sieht. Überzählige Zeige-Befehle führen dazu, daß die »Drop-Down-Menüs vom »Mauszeiger zerstört werden.

Außerhalb des aktiven Fensters sollte der Mauscursor **immer** die Form des Pfeiles ('gr_monumber' = 0) haben. Alles andere kann zu merkwürdigen Effekten führen. Die »*evnt_multi()*-Routine wird daher dazu benutzt, um ein Bereich anzugeben. Verläßt der »Mauszeiger diesen Bereich, so schaltet das Programm den Mauscursor auf Pfeilform um.

Eingabeparameter:

word gr_monumber — Dieser Parameter gibt die gewünschte Form des »Maus- zeigers an. Folgende Formen sind möglich:

- 0 - Pfeil
- 1 - Textcursor
- 2 - Biene (das "Busy"-Symbol)
- 3 - Hand mit zeigendem Finger
- 4 - Flache Hand
- 5 - Dünnes Fadenkreuz
- 6 - Dickes Fadenkreuz
- 7 - Umrahmtes Fadenkreuz
- 255 - Benutzer-definierte Mausform, deren Adresse in gr_mofaddr steht
- 256 - Verstecken des »Mauszeigers
- 257 - Sichtbarmachen des Mauscursors

Übrigens: zum Sichtbar- und Unsichtbarmachen des »Mauszeigers gibt es auch »Line-A-Befehle! Außerdem gibt es noch »VDI-Befehle für denselben Zweck.

word *gr_mofaddr — Zeiger auf ein Array, in dem die Daten für den zu zeigenden Mauscursor stehen. Der Wert ist nur von Belang, wenn 'gr_monumber' = 255 ist und wird ansonsten ignoriert (muß aber immer ein Langwort sein!).

```
contrl(0)   =  78
contrl(1)   =   1
contrl(3)   =   1

int_in(0)   =  gr_monumber

addr_in(0)  =  gr_mofaddr
```

Ausgabeparameter:

```
word gr_moreturn    0 = Ein Fehler ist passiert, n = alles in
                    Ordnung

contrl(2)   =   1
contrl(4)   =   0

int_out(0)  =  gr_moreturn
```

Graf_movebox

Zur »Graphics Library gehörende Routine

```
$48 = 72        gr_mreturn = graf_movebox(gr_mwidth, gr_mheight
word            gr_mreturn                gr_mwidth, gr_mheight;
                                          gr_msourcex, gr_msourcey,
word                                      gr_msourcex, gr_msourcey;
                                          gr_mdestx, gr_mdesty)
word                                      gr_mdestx, gr_mdesty;
```

Diese Routine zeichnet ein sich bewegendes Rechteck. Die Größe des Rechtecks ändert sich nicht. Meistens wird diese Routine nur benutzt, um dem Auge etwas zu bieten.

Eingabeparameter:

```
word gr_mwidth      Die Breite des Rechtecks

word gr_mheight     Die Höhe des Rechtecks

word gr_msourcex    Die X-Position des Rechtecks zu Beginn

word gr_msourcey    Die Y-Position des Rechtecks zu Beginn

word gr_mdestx      Die X-Position des Rechtecks nach Abschluß der
                    Bewegung

word gr_mdesty      Die Y-Position des Rechtecks nach Abschluß der
                    Bewegung
```

```
contrl(0)   =  72
contrl(1)   =   6
contrl(3)   =   0

int_in(0)   =  gr_mwidth
int_in(1)   =  gr_mheight
int_in(2)   =  gr_msourcex
int_in(3)   =  gr_msourcey
int_in(4)   =  gr_mdestx
int_in(5)   =  gr_mdesty
```

Ausgabeparameter:

```
word gr_moreturn   0 = Etwas ging schief, n = alles Ok

contrl(2)   =   1
contrl(4)   =   0

int_out(0)  =  gr_moreturn
```

Graf_rubberbox

Zur »Graphics Library gehörende Routine

```
$46 = 70        gr_greturn = graf_rubberbox(gr_rx, gr_ry,
word            gr_greturn                  gr_rx, gr_ry;
                                            gr_rminwidth,
word                                        gr_rminwidth,
                                            gr_minheight,
word                                        gr_minheight;
                                            &gr_rlastwidth,
word *                                      &gr_rlastwidth;
                                            &gr_rlastheight)
word *                                      &gr_rlastheight;
```

Diese Routine zeichnet ein Rechteck mit fester linker oberer Ecke. Die rechte untere Ecke kann vom Benutzer (solange der Mausknopf gedrückt wird) verschoben werden, wobei eine minimale Größe des Rechtecks vorgegeben ist (die maximale Größe wird durch die Grenzen des »Bildschirms definiert). Die Routine wird von »AES verwendet, wenn die Größe eines »Fensters vom Benutzer verändert wird (allerdings werden dabei zwei Rechtecke gezeichnet). Die Routine endet, sobald der Benutzer den Mausknopf losläßt und gibt die Größe des Rechtecks zurück.

Eingabeparameter:

```
word gr_rx              Die X-Koordinate des Rechtecks
```

word gr_ry	Die Y-Koordinate des Rechtecks
word gr_rminwidth	Die minimale Breite des Rechtecks
word gr_rminheight	Die minimale Höhe des Rechtecks
word *gr_rlastwidth word *gr_rlastheight	Adressen der Variablen für Rückgabewerte (s. u.)

```
contrl(0)   =  70
contrl(1)   =   4
contrl(3)   =   0

int_in(0)   =  gr_rx
int_in(1)   =  gr_ry
int_in(2)   =  gr_rminwidth
int_in(3)   =  gr_rminheight
```

Ausgabeparameter:

word gr_rreturn	0 = Ein Fehler ist passiert, n = alles Ok
word gr_rlastwidth	Die Breite des Rechtecks, nachdem der Benutzer den Mausknopf losgelassen hat
word gr_rlastheight	Die Höhe des Rechtecks, nachdem der Benutzer den Mausknopf losgelassen hat

```
contrl(2)   =  3
contrl(4)   =  0

int_out(0)  =  gr_rreturn
int_out(1)  =  gr_rlastwidth
int_out(2)  =  gr_rlastheight
```

Graf_shrinkbox

Zur »Graphics Library gehörende Routine

```
$4A = 74        gr_sreturn = graf_shrinkbox(gr_sfinx, gr_sfiny,
word            gr_sreturn                  gr_sfinx, gr_sfiny;
                                            gr_sfinwidth, gr_sfinheight,
word                                        gr_sfinwidth, gr_sfinheight;
                                            gr_sstx, gr_ssty,
word                                        gr_sstx, gr_ssty;
                                            gr_sstwidth, gr_sstheight)
word                                        gr_sstwidth, gr_sstheight;
```

Diese Routine zeichnet ein sich verkleinerndes Rechteck (das Gegenstück zu »*graf_growbox()*). Nett anzusehen, aber...

Eingabeparameter:

word gr_sfinx	Die X-Koordinate des Rechtecks nach der Verkleinerung
word gr_sfiny	Die Y-Koordinate des Rechtecks nach der Verkleinerung
word gr_sfinwidth	Die Breite des Rechtecks nach der Verkleinerung
word gr_sfinheight	Die Höhe des Rechtecks nach der Verkleinerung
word gr_sstx	Die X-Koordinate des Rechtecks vorher
word gr_ssty	Die X-Koordinate des Rechtecks vorher
word gr_sstwidth	Die Breite des Rechtecks vorher
word gr_sstheight	Die Höhe des Rechtecks vorher

```
contrl(0)   =  74
contrl(1)   =   8
contrl(3)   =   0

int_in(0)   =  gr_sfinx
int_in(1)   =  gr_sfiny
int_in(2)   =  gr_sfinwidth
int_in(3)   =  gr_sfinheight
int_in(4)   =  gr_sstx
int_in(5)   =  gr_ssty
int_in(6)   =  gr_sstwidth
int_in(7)   =  gr_sstheight
```

Ausgabeparameter:

word gr_sreturn 0 = Ein Fehler ist passiert, n = kein Fehler

```
contrl(2)   =   1
contrl(4)   =   0

int_out(0)  =  gr_sreturn
```

Graf_slidebox

Zur »Graphics Library gehörende Routine

```
$4C = 76        gr_slreturn = graf_slidebox(gr_slptree, gr_slparent,
word            gr_slreturn,                            gr_slparent;
word *                                  gr_slptree
                                        gr_slobject, gr_slvh)
word                                    gr_slobject, gr_slvh;
```

Diese Routine ähnelt »*graf_dragbox()*. Auch hier wird ein Rechteck in den Grenzen eines anderen bewegt. Hier allerdings werden nicht die Bereiche und die Größe der zu bewegenden Rechtecke angegeben, sondern ein Elternobjekt, das die äußeren Grenzen festlegt, und ein Kindobjekt eines »Objektbaums. Das Kindobjekt wird innerhalb der Grenzen des Elternobjekts mit der »Maus bewegt. Beide »Objekte **müssen** im selben »Objektbaum enthalten sein. Die zurückgegebene Position bezieht sich auch nicht auf den »Bildschirm (wie bei »*graf_dragbox()*), sondern wird relativ zum Elternobjekt angegeben. Die Routine kehrt zurück, wenn der Benutzer den Mausknopf losläßt.

Die Rollbalken und Schieber eines »Fensters werden mit dieser Routine verschoben (was an den Rückgabewerten leicht zu erkennen ist).

Eingabeparameter:

word *gr_slptree	Die Adresse des »Objektbaums, der das Elternobjekt und das Kindobjekt enthält
word gr_slparent	Die Indexnummer des Elternobjekts im obigen »Objektbaum
word gr_slobject	Die Indexnummer des zu bewegenden »Objekts, dessen Elternobjekt mit 'gr_slparent' angegeben wurde
word gr_slv	Dieses Flag gibt die Bewegungsrichtung an: 0 = horizontale Bewegung 1 = vertikale Bewegung

```
contrl(0)   =  76
contrl(1)   =   3
contrl(3)   =   1

int_in(0)   =  gr_slparent
int_in(1)   =  gr_slobject
int_in(2)   =  gr_slvh
```

addr_in(0) = gr_slptree

Ausgabeparameter:

word gr_slreturn Die neue Position des Zentrums des Kindobjekts relativ zum Elternobjekt - ein Wert zwischen 0 und 1000. Bei horizontaler Bewegung bedeutet 0 links und 1000 ganz rechts, bei vertikaler Bewegung entspricht 0 oben und 1000 unten.

contrl(2) = 1
contrl(4) = 0

int_out(0) = gr_slreturn

Graf_watchbox

Zur »Graphics Library gehörende Routine

```
$4B = 75        gr_wreturn = graf_watchbox(gr_wptree, gr_wobject,
word            gr_wreturn,                            gr_wobject;
word *                                     gr_wptree;
                                           gr_winstate, gr_woutstate)
word                                       gr_winstate, gr_woutstate;
```

Diese Routine ändert den Status eines »Objekts in Abhängigkeit davon, ob sich der Mauszeiger innerhalb oder außerhalb der Objektgrenzen befindet. Zum Status von Objekten bitte unter dem Stichwort »Objekt nachsehen!

Das »Objekt selbst wird als ein Rechteck angesehen. Die Routine wird aufgerufen, wenn der Mausknopf gedrückt ist, und sie kehrt zurück, wenn der Benutzer den Mausknopf losläßt.

Eingabeparameter:

word *gr_wptree Die Adresse des Objektbaums, der das rechtekkige »Objekt enthält

word gr_wobject Die Indexnummer des »Objekts im Objektbaum

word gr_winstate Der Zustand des »Objekts, wenn sich der Mauszeiger innerhalb dessen Grenzen befindet

word gr_woutstate Der Zustand des »Objekts, wenn sich der Mauszeiger außerhalb dessen Grenzen befindet Die Variablen 'gr_winstate' und 'gr_woutstate' können folgende Werte annehmen:

Bit	zugehöriger Zustand
0	SELECTED = ausgewählt
1	CROSSED = durchkreuzt
2	CHECKED = mit Haken
3	DISABLED = desaktiviert
4	OUTLINED = umrahmt
5	SHADOWED = mit Schatten

Ein Wert von NULL entspricht dem Zustand 'NORMAL'. Die Zustände können durch Setzen der entsprechenden Bits auch gemischt werden. Die Bedeutung der Zustände ist unter »Objekt erklärt.

```
contrl(0)   =  75
contrl(1)   =   4
contrl(3)   =   1

int_in(0)   =  [kein Wert - reserviert!]
int_in(1)   =  gr_wobject
int_in(2)   =  gr_winstate
int_in(3)   =  gr_woutstate

addr_in(0)  =  gr_wptree
```

Ausgabeparameter:

word gr_wreturn Gibt die Position des Mauszeigers an, wenn der Benutzer den Mausknopf losläßt, wobei eine NULL bedeutet, daß dies **außerhalb** des »Objekts geschah, und eine Eins anzeigt, daß dies **innerhalb** des »Objekts passierte

```
contrl(2)   =   1
contrl(4)   =   0

int_out(0)  =  gr_wreturn
```

Handle
Engl., wörtlich: Griff. Als H. bezeichnet man Zahlen, die den Zugriff auf etwas regeln, also beispielsweise auf Dateien oder »Fenster (siehe »Window handle). Mit anderen Worten: es handelt sich um eine Kennummer.

Handlenummer
Siehe »Handle

Hierarchisches Filesystem
Hierbei handelt es sich um ein strukturiertes Filesystem, das folgenden Aufbau hat: Jeder Massenspeicher besitzt sein eigenes Filesystem. Dieses ist baumartig aufgebaut, wobei die Bäume in der Informatik ja bekanntlich von oben nach unten wachsen. Die Wurzel dieses Baumes wird als Rootdirectory bezeichnet. Dieses Directory enthält sowohl normale Files wie Programme und/oder Datenfiles als auch eventuell weitere sogenannte Subdirectories. Diese können wiederum Files und/oder weitere Subdirectories enthalten. Um nun ein einzelnes File anzusprechen, muß außer dem Filenamen auch noch der Zugriffspfad (»Pfad) zu dem Directory bekannt sein, in dem sich dieses File befindet. Das Betriebssystem stellt Routinen zur Verfügung, mit deren Hilfe man sich bequem durch diesen Baum aus Subdirectories hindurchhangeln kann.

I/O-Chips

Darunter versteht man Integrierte Schaltkreise, die sich um die Ein- und Ausgabe von Daten kümmern. Sie verbinden die CPU mit der Außenwelt, stellen also die Verbindung zwischen einer Schnittstelle und der CPU her. Diese Chips sind meist für die Steuerung von bestimmten Schnittstellen oder Peripheriegeräten spezialisiert. Im Atari ST stehen davon zur Verfügung: ein Multifunktionsbaustein »MC68901, ein »Soundchip des Typs YM-2149, zwei serielle Controller des Typs »ACIA6850 und ein Floppy Controller vom Typ »FDC1772. Die Steuerung des Videobildschirms teilen sich mehrere IC's, so daß man hier kaum von I/O-Chips sprechen kann. Der »Tastaturprozessor ist, wie sein Name schon sagt, kein I/O-Chip, sondern ein eigenständiger Prozessor, der der CPU des Hauptrechners eine Menge Arbeit abnimmt.

ICONBLK

Hierbei handelt es sich um eine Struktur, die Informationen über Objekte vom Typ »Icon enthält. Aufbau, Sinn und Zweck der Struktur ist unter dem Stichwort »Objekt näher erklärt.

Icon

Ein I. ist ein grafisch dargestelltes Bild, das durch Anklicken (mit der Maus) eine bestimmte Funktion ausführt. Auf dem »Desktop sind das beispielsweise die Laufwerke. Klickt man diese doppelt an, so stellt der Computer auf dem Monitor das »Directory der angeklickten Diskettenstation dar (auf Wunsch wird wieder jede »Datei als ein I. dargestellt).

Der Aufbau von Icons ist unter dem Stichwort »Objekt genauer beschrieben.

Info-Zeile

Bezeichnung für einen Bestandteil von »Fenstern. Siehe »Window

Intel-Format

Beim I. sind High und Low Byte vertauscht, d.h. es folgt erst das Low Byte und dann das High Byte. Das I. wird bei der Diskettenverwaltung benutzt.

Interrupt

Engl.: Unterbrechung. Damit bezeichnet man bei Computern eine Anforderung eines »I/O-Chips an die CPU, sich um ihn zu kümmern, z.B. um ein gerade von einer Schnittstelle eingetroffenes Zeichen abzuholen. Zu diesem Zweck muß natürlich das gerade laufende Pro-

gramm unterbrochen werden. Ein I. wird im Atari ST folgendermaßen bearbeitet: Der »I/O-Chip, der die CPU benötigt, wechselt den Pegel an einem seiner Statuspins. Weitere Chips sorgen dafür, daß dieses Signal als I.anforderung auf einem der Pegel 1 bis 7 an die CPU gemeldet wird. Im Atari ST werden leider hardwaremäßig nur die Pegel 2, 4 und 6 ausgewertet. Das Mikroprogramm der CPU vergleicht diesen Pegel mit den Bits 8 bis 10 des Statusregisters. Wenn der Pegel über dem Wert dieser Bits liegt, wird der aktuelle Befehl zuende bearbeitet. Dann wird der I.pegel in diese Bits übernommen, so daß ein weiterer I. auf diesem Pegel die CPU nicht noch einmal unterbrechen kann. Entsprechend dem angelegten Pegel und dem Signal am Pin DTACK wird entwder der zugehörige Autovektor oder Non-Autovektor geholt. Die CPU legt den Wert des PC's auf den Stack und packt das Statusregister gleich hinterher. Dann wird im Statusregister das »Supervisorbit gesetzt und das »Tracebit gelöscht. Dann wird die zu dem Vektor gehörende I.routine ausgeführt. Diese Routine muß mit einem RTE-Befehl enden. Dieser Befehl sorgt dafür, daß der CPU interne I.level wieder auf den alten Wert gesetzt wird und daß das Statusregister und der PC vom Stapel geholt werden. Dann wird das unterbrochene Programm ganz normal weitergeführt.

Interruptstruktur

Die I. gibt an, welche »Interrupts von welchen Chips ausgelöst werden können und mit welcher Priorität diese bearbeitet werden. Im Atari ST erzeugt der Videocontroller jedesmal einen Interrupt des Pegels 2, wenn der Elektronenstrahl auf seinem Rückweg vom rechten zum linken Rand des Bildschirms dunkelgetastet wird. Dieser Interrupt tritt beim Monochrommonitor alle 50 µs und bei anderen Monitoren alle 64 µs auf. Er wird normalerweise von der CPU nicht bearbeitet, weil deren Interruptmaske den Wert drei hat. Außerdem löst der Videocontroller auch jedesmal einen Interrupt aus, wenn der Elektronenstrahl auf seinem Weg von unteren zum oberen Rand des Schirmes dunkelgetastet wird. Dieser Interrupt liegt auf Level 4. Er führt jedesmal zur Ausführung der »V-Blank Routine.

Auf dem Level 6 liegen die Interrupts des Multifunktionsbausteins »MC68901. Im Gegensatz zu den eben genannten Interrupts handelt es sich bei diesen um Non-Autointerrupts. Diese Interrupts haben die höchste Priorität im Atari ST und benutzen die Vektoren von 64 bei Adresse $000100 bis 79 bei Adresse $00013C. Die Zuordnung zu den einzelnen Interruptquellen im MC68901 finden Sie unter dem Stichwort »Exceptions.

Justiz

Die J. rückt Ihnen auf den Pelz, wenn Sie sich bei der Anfertigung von »Raubkopien erwischen lassen. Siehe »Raubkopie.

Keyboard-Event
Engl., zu deutsch: Tastatur-Ereignis. Siehe »Event.

Kindobjekt
Siehe »Objekt

Klemmbrett
siehe »Clipboard

Kommandointerpreter
Ein K. ist eine textorientierte Benutzeroberfläche. Der Benutzer gibt leicht merkbare Befehle ein (sofern es sich nicht um das Betriebssystem UNIX™ handelt), die vom K. in Aufrufe der entsprechenden Routinen des Betriebssystems umgewandelt werden. Der K. überwacht dann auch noch die Ausführung dieser Aufrufe. Zusätzlich enthalten viele K. weitere Hilfsroutinen, die über die vom Betriebssystem zur Verfügung gestellten hinausgehen. Ein weiterer Vorteil eines K.s ist dessen integrierter Batch-Prozessor. Dieser bearbeitet Batchdateien genauso, als ob der Benutzer ein Eingabe am Terminal machen würde.

Library

Engl., auf deutsch: Bibliothek. Als L. bezeichnet man in der Computer-Programmierung eine Sammlung von Routinen und Funktionen, die logisch zusammengehören.

Line-A

Unter Line-A versteht man zunächst einmal »Opcodes des MC 68000 Prozessors, deren höchste vier Bit den Wert %1010 (=$A) haben. Alle diese »Opcodes ($Axxx) stellen Befehle dar, die der Prozessor nicht verarbeiten kann. Wenn er auf einen solchen Befehl trifft, führt er eine »Exception aus. Dabei handelt es sich um einen indirekten Sprung an die Adresse $000028.

Beim Atari ST steht dort ein Vektor auf eine Routine, die 16 verschiedene Grafikbefehle bearbeitet. Diese Befehle werden durch die Bits 0-3 im »Opcode unterschieden. Alle Befehle benötigen zusätzliche Parameter, die vor Benutzung des Line-A Befehls in den sogenannten Line-A Variablen hinterlegt werden müssen. Es folgen zunächst die Variablen, dann die Befehle. Die Startadresse der Line-A Variablen ist $00293A, wenn sich das Betriebssystem im ROM befindet. Um auf eine bestimmte Variable zugreifen zu können, muß der jeweils angegebene Offset zu dieser Startadresse hinzuaddiert werden. Die Namen der Variablen entstammen dem Entwicklungspaket. Alle Variablennamen beginnen mit einem Unterstrich. Diese Konvention gilt für alle Variablen, die direkt vom Betriebssystem benutzt werden.

Offset	Länge	Name	Bedeutung
$00=0	word	_v_planes	Anzahl der Bildebenen (»Bildschirmspeicher)
$02=2	word	_v_lin_wr	Anzahl Bytes je Zeile.
$04=4	*word	_CONTROL	Zeiger auf das CONTROL-Array
$06=6	*word	_INTIN	Zeiger auf das INTIN-Array
$0C=12	*word	_PTSIN	Zeiger auf das PTSIN-Array
$10=16	*word	_INTOUT	Zeiger auf das INTOUT-Array
$14=20	*word	_PTSOUT	Zeiger auf das PTSOUT-Array
$18=24	word	_FG_BP_0	Farbwert für Bildebene 0 (»Bildschirmspeicher). Es wird nur Bit 0 benutzt.
$1A=26	word	_FG_BP_1	Farbwert für Bildebene 1. Es wird nur Bit Bit 0 benutzt.
$1C=28	word	_FG_BP_2	Farbwert für Bildebene 2. Es wird nur Bit Bit 0 benutzt.
$1E=20	word	_FG_BP_3	Farbwert für Bildebene 3. Es wird nur Bit 0 benutzt.
$20=32	word	_LSTLIN	Muß den Wert -1=$FFFF haben.
$22=34	word	_LN_MASK	Linienmuster für $A003-Befehl.

$24=36	word	_WRT_MODE	Zeichenmodus. Es gibt vier mögliche Werte 0=Neu schreiben, Hintergrund löschen. 1=Transparent, ODER-Verknüpfung mit dem Hintergrund. 2=Inverser Modus; EXOR-Verknüpfung mit dem Hintergrund. 3=Invers-Transparent; Schreibmaske invertieren, dann ODER-Verknüpfung mit dem Hintergrund.
$26=38	word	_X1	X-Koordinate des ersten Punktes.
$28=40	word	_Y1	Y-Koordinate des ersten Punktes.
$2A=42	word	_X2	X-Koordinate des zweiten Punktes.
$2C=44	word	_Y2	Y-Koordinate des zweiten Punktes.
$2E=46	*word	_PATPTR	Zeiger auf das Füllmuster für den $A004-Befehl.
$32=50	word	_PATNBR	Anzahl der Elemente im Füllmusterarray-1.
$34=52	word	_MULTIFL	Regelt die Benutzung des Füllmusters. 0=gilt nur für eine Bildebene. 1=gilt für alle Bildebenen.
$36=54	word	_CLIP	0=kein Clipping, 1=Clipping. »$A005-Befehl.
$38=56	word	_XMN_CLIP	Legt die X-Koordinate der linken oberen Ecke für das Clipping-Rechteck fest.
$3A=58	word	_YMN_CLIP	Legt die zugehörige Y-Koordinate fest.
$3C=60	word	_XMX_CLIP	Legt die X-Koordinate der rechten unteren Ecke für das Clipping-Rechteck fest.
$3E=62	word	_YMX_CLIP	Legt die zugehörige Y-Koordinate fest.
$40=64	word	_XACC_DDA	Muß vor Benutzung des $A008-Befehls auf $8000 gesetzt werden.
$42=66	word	_DDA_INC	Legt den Verkleinerungs- bzw. Vergrößerungsfaktor bei Textausgaben fest.
$44=68	word	_T_SCLSTS	Gibt an, ob verkleinert oder vergrößert werden soll: 0=verkleinern, 1=vergrößern.
$46=70	word	_M_STATUS	Schriftstatus. 0=Normaler Font (»Font), 1=Font für Proportionalschrift.
$48=72	word	_SOURCEX	Enthält die X-Koordinate (Offset) eines Zeichens im Font und ist folgendermaßen erhältlich: Man subtrahiert den Wert FIRST (Byte 35,36 des Fontheaders) vom ASCII-Wert des Zeichens. Das Resultat dient als Offset in die Character Offset Table (»Font). Der dort stehende Wert ist dann _SOURCEX.
$4A=74	word	_SOURCEY	Enthält die Y-Koordinate eines Zeichens im Font und ist immer Null (?!).
$4C=76	word	_DESTX	Gewünschte X-Position des Zeichens auf dem Bildschirm.
$4E=78	word	_DESTY	Gewünschte Y-Position des Zeichens.

$50=80	word	_DELX	Breite des Zeichens. Dieser Wert ist ebenfalls aus der Character Offset Table erhältlich. Man muß dazu auch _SOURCEX für das nächste ASCII-Zeichen berechnen und obigen Wert für _SOURCEX subtrahieren.
$52=82	word	_DELY	Höhe des Zeichens. Dieser Wert kann den Bytes 81,82 des Fontheaders entnommen werden.
$54=84	*word	_FBASE	Zeiger auf den Beginn des Fontheaders.
$56=88	word	_FWIDTH	Summe der Längen aller Zeichen im Font.
$5A=90	word	_STYLE	Einzelne Bits in diesem Wort erzeugen spezielle Effekte; näheres beim $A008-Befehl.
$5C=92	word	_LITEMASK	Maske für das Schattieren. Steht im Fontheader, üblicherweise $5555.
$5E=94	word	_SKEWMASK	Maske für Kursivschrift. Steht im Fontheader, üblicherweise $5555.
$60=96	word	_WEIGHT	Anzahl an Pixeln, um die das Zeichen verbreitert werden soll.
$62=98	word	_R_OFF	Um diesen Wert wird die Zelle eines Zeichens nach rechts erweitert, wenn das Zeichen kursiv ausgegeben werden soll.
$64=100	word	_L_OFF	Um diesen Wert wird die Zelle nach links erweitert.
$66=102	word	_SCALE	0=weder Vergrößerung noch Verkleinerung.
$68=104	word	_CHUP	Winkelangabe für Zeichenrotation. Es funktionieren nur 90°-Winkel. Dabei sind folgende Werte möglich: 0 = Zeichen wird normal dargestellt 900 = Zeichen ist um 90° nach rechts gedreht. 1800 = Zeichen steht auf dem Kopf, es ist um 180° gedreht. 2700 = Zeichen ist um 270° gedreht.
$6A=106	word	_TEXT_FG	Farbe des Zeichens selbst.
$6C=108	*word	_SCRTCHP	Zeiger auf einen Zwischenpuffer, der für die Texteffekte benötigt wird.
$70=112	word	_SCRPT2	Offset für den Puffer für spezielle Effekte vom Start des Zwischenpuffers aus gesehen.
$72=114	word	_TEXT_BG	Hintergrundfarbe der Zelle eines Zeichens.

Nachdem die zu einem bestimmten Line-A Befehl gehörenden Variablen versorgt sind, kann der Befehl aufgerufen werden. Es folgen die 16 möglichen Befehle. Jeder dieser Befehle arbeitet in allen drei Auflösungsmodi des Atari ST.

$A000 - Start Line-A
Um überhaupt mit den Line-A Befehlen arbeiten zu können, muß zuerst dieser Befehl benutzt werden. Er liefert in den Registern D0 und A0 die Startadresse der Line-A Variablen zurück. Ein guter Programmierer verläßt sich nicht darauf, daß ATARI ihre Variablen immer an derselben Stelle läßt sondern benutzt diesen Befehl, um die Startadresse zu erhalten. Das Register A1 liefert einen Zeiger auf eine Tabelle, die die Startadressen der drei System Zeichensätze enthält. Das Register A2 zeigt auf eine Tabelle, die die Startadressen der 16 Line-A Befehle enthält. Außer den genannten werden noch die Register D1 und D2 verändert.

$A001 - Set Pixel
Mit diesem Befehl wird ein einzelner Punkt auf dem Bildschirm gesetzt. Dessen Koordinaten müssen im PTSIN-Array übergeben werden. PTSIN(0) enthält die X-Koordinate und PTSIN(1) die Y-Koordinate. Die Farbe des Punktes wird in INTIN(0) übergeben. Je nach Grafikmodus werden ein, zwei oder vier Bits benutzt. Liegt der X-Wert außerhalb des erlaubten Bereiches, dann wird er solange um die maximale Bildschirmbreite verringert und gleichzeitig der Y-Wert inkrementiert, bis er im erlaubten Bereich liegt. Liegt der Y-Wert außerhalb des erlaubten Bereiches, erfolgt keine Ausgabe. Dieser Befehl verändert die Register A0 bis A2 und D1.

$A002 - Get Color
Dieser Befehl liefert die Farbe eines Punktes im Register D0 zurück. Die Koordinaten des Punktes müssen wiederum im PTSIN-Array übergeben werden. In PTSIN(0) die X-Koordinate und in PTSIN(1) die Y-Koordinate. Es werden ebenfalls die Register A0 bis A2 und D1 verändert.

$A003 - Draw Line
Hiermit wird eine beliebige Linie zwischen zwei Punkten gezogen. Alle benötigten Parameter werden in den Line-A Variablen übergeben. Die gewünschte Farbe wird in den Variablen _FG_BP_x (Offsets 24-30) übergeben, wobei in jeder dieser Variablen nur das Bit 0 eine Bedeutung hat. Alle vier Bits zusammen bestimmen eine von maximal 16 möglichen Farben. Spätestens bei Benutzung dieses Befehls muß die Variable _LSTLIN (Offset 32) auf den Wert $FFFF gesetzt werden. Die Variable _LN_MASK (Offset 34) erhält einen Wert, der das Linienmuster bitweise codiert. Ist dieser Wert z.B. $6666, dann wird in der Linie immer ein Punkt freigelassen, dann werden zwei gesetzt, usw. Die Variable _WRT_MOD (Offset 36) erhält einen von vier möglichen Zeichenmodi. Dabei bedeutet 0, daß alle unter der Linie existierenden Punkte vor dem Zeichnen gelöscht werden. 1 bedeutet, daß alle unter der Linie liegenden Punkte bleiben, d.h. es wird eine ODER-Verknüpfung mit dem bereits vorhandenen Muster durchgeführt. Der Wert 2 bedeutet, daß eine Exklusiv-ODER-Verknüpfung mit dem vorhandenen Muster durch-

geführt wird. Der letzte mögliche Wert drei führt zu einer Invertierung der Maske und anschließender ODER-Verknüpfung mit dem Muster.

Die Koordinaten vom Start- und Endpunkt der Linie werden in den Variablen _X1 (Offset 38) bis _Y2 (Offset 44) übergeben. Liegt einer der beiden Punkte außerhalb des Bildschirmbereiches, wird trotzdem weitergezeichnet!!! Hier ist also höchste Vorsicht geboten, da dadurch andere RAM-Bereiche überschrieben werden. Es werden die Register D0, D1 und A0-A2 verändert.

$A004 - Draw horizontal line

Mit diesem Befehl können nur horizontale Linien gezogen werden. Deshalb werden zwei X- und nur eine Y-Koordinate benötigt. Diese werden in den Variablen _X1 (Offset 38), _Y1 (Offset 40) und in _X2 (Offset 42) übergeben. Die Farbe wird wieder in den Variablen _FG_BP_x (Offsets 24-30) übergeben. In _WRT_MOD (Offset 36) steht wieder der Zeichenmodus (siehe $A003-Befehl).

Dieser Befehl wird von den Füllroutinen $A005 und $A006 aufgerufen. Es ist möglich, jeder Zeile ein eigenes Füllwort zuzuordnen. Diese Musterworte werden in einem eindimensionalen Array abgelegt. In der Variablen _PATPTR (Offset 46) muß ein Zeiger auf dieses Array hinterlegt werden und in der Variablen _PATNBR (Offset 50) muß die Anzahl minus Eins der Elemente dieses Arrays stehen. Es werden die Register D0, D1 und A0-A2 verändert.

$A005 - Filled rectangle

Mit diesem Befehl wird ein mit einem Muster gefülltes Rechteck gezeichnet. Dieses Rechteck wird durch seine linke obere und seine rechte untere Ecke festgelegt. Die Koordinaten dieser beiden Punkte werden wieder in den Variablen _X1 (Offset 38) bis _Y2 (Offset 44), die Farbe in den Variablen _FG_BP_x (Offsets 24 bis 30) und der Zeichenmodus in _WRT_MOD (Offset 36) hinterlegt. Hinzu kommen wie beim $A004-Befehl die Variablen für das Füllmuster: der Zeiger auf das Füllmusterarray in _PATPTR (Offset 46) und die Anzahl der Elemente minus Eins in _PATNBR (Offset 50). Zusätzlich können bei diesem Befehl auch noch die sogenannten Clippingvariabeln ausgewertet werden. Diese bestehen aus fünf Variablen. Die Variable _CLIP (Offset 54) bestimmt, ob die Koordinaten ausgewertet werden sollen. Um das Clipping zu benutzen, muß diese einen Wert ungleich Null haben. In den Variablen _XMN_CLIP und _YMN_CLIP (Offsets 56 und 58) stehen die Koordinaten der linken oberen Ecke eines Rechtecks, und in den Variablen _XMX_CLIP und _YMX_CLIP (Offsets 60 und 62) stehen die Koordinaten der rechten unteren Ecke dieses Rechtecks. Dieses Rechteck legt nun den Bereich des Bildschirmes fest, in dem das gefüllte Rechteck gezeichnet werden darf.

$A006 - Filled polygon

Mit diesem Befehl wird für eine in der Variablen _Y1 (Offset 40) übergebene Linie untersucht, ob Teile dieser Zeile innerhalb eines Polygons liegen. Die einzelnen Punkte dieses Polygons müssen im PTSIN-Array hinterlegt werden. Die Variable _PTSIN (Offset 12) zeigt auf dieses Array. Dabei ist folgendes zu beachten: Zuerst wird die X-Koordinate des ersten Punktes eingetragen, dann dessen Y-Koordinate. Die Koordinaten des letzten Punktes im Array müssen wieder gleich denen des ersten sein, damit überhaupt ein geschlossenes Polygon vorliegt. In dem Element CONTRL(0) muß die Anzahl der echten Punkte im Polygon hinterlegt werden. Dabei wird der letzte Punkt nicht noch einmal gezählt! Da mit diesem Befehl nur für eine Y-Koordinate überprüft wird, ob ein Teil der Linie gezeichnet werden soll, muß der Programmierer selbst dafür sorgen, daß das gesamte Polygon gezeichnet wird. Dies geschieht am besten, indem zuerst der kleinste und der größte Wert der Y-Koordinaten im PTSIN-Array gefunden werden. Dann wird die Variable _Y1 (Offset 38) auf den kleinsten Wert gesetzt. Danach wird in einer Schleife der $A006-Befehl ausgeführt und die Variable wird inkrementiert, bis sie den größten Wert erreicht hat.

Das Füllmuster wird wie beim $A005-Befehl übergeben. Außerdem sind die Variablen für die Farbe, den Zeichenmodus und das Clipping zu belegen. Es werden die Register D0, D1 und A0-A2 verändert.

$A007 - Bitblock transfer (BITBLT)

Mit diesem Befehl werden Bitblöcke auf den Bildschirm übertragen. Der Befehl wird von den Befehlen $A008 und $A00E aufgerufen. Im Register A6 muß ein Zeiger auf eine Parametertabelle übergeben werden, die folgenden Aufbau hat:

Offset	Länge	Name	Bedeutung
$00=0	word	_f_wd	Breite der Form in Pixeln
$02=2	word	_f_ht	Höhe der Form in Pixeln
$04=4	word	_plane_ct	Anzahl der Planes des Bildschirms.
$06=6	word	_fg_col	Fordergrundfarbe.
$08=8	word	_bg_col	Hintergrundfarbe.
$0A=10	long	_ob_tab	Logischer Operator für Bitverknüpfung.
$0E=14	word	_s_xmin	X-Koordinate der Quellform
$10=16	word	_s_ymin	Y-Koordinate der Quellform
$12=18	long	_s_form	Basisadresse der Quellform
$16=22	word	_s_nxwd	Offset zum nächsten Wort der Quellform in Bytes.
$18=24	word	_s_nxln	Offset zur nächsten Linie der Quellform.
$1A=26	word	_s_nxpl	Offset zur nächsten Plane (immer 2).
$1C=28	word	_d_xmin	X-Koordinate der Zielform
$1E=30	word	_d_ymin	Y-Koordinate der Zielform
$20=32	long	_d_form	Basisadresse der Zielform

$24=36	word	_d_nxwd	Offset zum nächsten Wort der Zielform in Bytes.
$26=38	word	_d_nxln	Offset zur nächsten Linie der Zielform.
$28=40	word	_d_nxpl	Offset zur nächsten Plane (immer 2).
$2A=42	long	_p_addr	Zeiger auf ein Füllmuster ($000000, wenn nicht benutzt).
$2E=46	word	_p_nxln	Offset zur nächsten Linie des Füllmusters
$30=48	word	_p_nxpl	Offset zur nächsten Plane des Füllmusters
$32=50	word	_p_mask	Indexmaske für ds Füllmuster.
$34=52	word	filler	12 Worte für spätere Erweiterungen.

$A008 - Textblock transfer (TEXTBLT)
Mit diesem Befehl wird ein beliebiges Zeichen aus einem der »Fonts auf eine beliebige Bildschirmposition übertragen. Das Zeichen kann in seiner Form verändert werden. Damit wird Kursivschrift, Fettdruck, Unterstreichen, Breitschrift möglich. Außerdem kann das Zeichen gedreht und vergrößert werden (das Verkleinern funktioniert leider nicht). Es werden alle Variablen ab _CLIP (Offset 54) ausgewertet.

$A009 - Show mouse
Mit diesem Befehl kann der Mauszeiger sichtbar gemacht werden und den Bewegungen mit der Maus folgen. Die Variable CONTROL(1) sollte gelöscht werden und CONTROL(3) sollte den Wert 1 erhalten. Das Betriebssystem zählt die Aufrufe zum Löschen des Cursors ($A00A) mit. Steht nun in INTIN(0) ein Wert ungleich Null, dann muß dieser Befehl sooft ausgeführt werden, bis dieser Zähler wieder gelöscht ist. Es werden die Register D0, D1 und A0-A2 verändert.

$A00A - Hide mouse
Mit diesem Befehl kann der Mauszeiger ausgeschaltet werden. Die Anzahl an Aufrufen dieses Befehls wird mitgezählt. Es werden die Register D0, D1 und A0-A2 verändert.

$A00B - Transform mouse
Hiermit kann das Aussehen des Mauszeigers verändert werden. Die benötigten Daten werden in INTIN-Array übergeben. In INTIN(3) steht die Farbe der Maske des Zeigers. Diese ist üblicherweise Null. In INTIN(4) steht die Farbe des Mauszeigers selbst. Diese ist bei einem Schwarzweiß-Monitor gleich Eins. Die Elemente INTIN(5) bis INTIN(20) enthalten die 16 Worte, die für die Maske benötigt werden und die Elemente INTIN(21) bis INTIN(36) enthalten die 16 Worte, die für den Mauszeiger benötigt werden. Es werden die Register D0, D1 und A0-A2 verändert.

$A00C - Undraw sprite
Dieser Befehl löscht ein auf dem Bildschirm stehendes Sprite. Zu diesem Zweck muß im Register A2 ein Zeiger auf den Zwischenpuffer

übergeben werden, in dem die Bildschirmdaten beim Zeichnen des Sprites hinterlegt wurden. Es ist somit durchaus sinnvoll, sich den Zeiger auf den Zwischenpuffer beim Spritezeichnen zu merken. Zeigt das Register A2 nicht auf diesen Puffer, dann können die interessantesten Effekte auftreten. Außer dem Register A2 werden ebenfalls wieder die Register D0, D1 und A0, A1 verändert.

$A00D - Draw sprite

Diese Routine bringt ein Sprite auf den Bildschirm. Die linke obere Ecke des Sprites, das übriegens eine Grösse von 16*16 Pixeln hat, wird durch die in den Registern D0 (X) und D1 (Y) übergebenen Koordinaten festgelegt. Das Register A2 zeigt auf den benötigten Zwischenpuffer, in den der Bildschirminhalt gesichert wird. Dieser Puffer muß eine Größe von 5 Worten für allgemeine Daten und von weiteren 32 Worten für jede benutzte Plane (Bildschirmebene) haben. Bei monochromem Bildschirm ist die Puffergrösse somit 37 Worte, bei vierfarbigem (640*200 Punkte) 2*32+5=69 Worte und bei 16 farbigem (320*200 Punkte) 4*32+5=133 Worte groß. Das Register A0 zeigt auf einen sogenannten Sprite Definitions Block. Dieser hat eine Größe von 37 Worten und folgenden Aufbau:

Wort 0 - Dieser Wert wird zur X-Koordinate in D0 hinzuaddiert.
Wort 1 - Dieser Wert wird zur Y-Koordinate in D1 hinzuaddiert.
Wort 2 - Beschreibt das Zeichenformat: 0=VDI, 1=XOR-Format.
Wort 3 - Farbe der Maske.
Wort 4 - Farbe des Sprites selbst.
Wort 5 - Aussehen der Maske in Zeile 0.
Wort 6 - Aussehen des Sprites in Zeile 0.
Wort 7 - Aussehen der Maske in Zeile 1.
Wort 8 - Aussehen des Sprites in Zeile 1.
Wort 9 - Aussehen der Maske in Zeile 2.
Wort 10 - Aussehen des Sprites in Zeile 2.
usw.
Es werden die Register D0, D1, A0-A2 und auch A6 verändert!

$A00E - Copy raster form

Mit diesem Befehl werden rechteckige Teile des Bildschirms in den Speicher übertragen, von dort zurückgeholt oder auf dem Bildschirm selbst verschoben. Ein Raster ist ein rechteckiger Teil des Bildschirms, der auf einer Wortgrenze beginnen muß. Dieser Routine müssen Zeiger auf zwei "Memory Form Definition blocks" übergeben werden. Einer dieser Blöcke beschreibt das Quellraster und der andere das Zielraster. Diese Blöcke (MFDB's) haben folgenden Aufbau:

Wort 0+1	Enthalten einen Zeiger auf das erste Wort des Rasters, das von diesem Block beschrieben wird.
Wort 2	Legt die Breite des Rasters in Pixeln fest.
Wort 3	Legt dessen Höhe in Pixeln fest.
Wort 4	Legt die Breite des Rasters in Worten fest. Dieser Wert ist: INT(Wort_2+15)
Wort 5	Muß den Wert 1 für das VDI-Format enthalten.
Wort 6	Gibt die Anzahl an Bildebenen (Planes) an, aus denen das Raster besteht.
Wort 7-9	Sind reserviert für spätere Erweiterungen und sollten auf Null gesetzt werden.

Leider ist noch nicht bekannt, wie diese Zeiger übergeben werden müssen, und ob noch weitere Parameter zu versorgen sind.

$A00F - Contour fill

Diese Routine füllt eine durch den Bildschirmrand oder durch eine andere Farbe begrenzte Fläche aus. In PTSIN(0) wird die X-Koordinate des Startpunktes übergeben und in PTSIN(1) dessen Y-Koordinate. In INTIN(0) wird ein Farbwert übergeben, der die Größe der zu füllenden Fläche festlegt. Ist dieser Wert positiv, wird eine Fläche ausgefüllt, deren Rand durch Punkte bestimmt ist, die die Farbe INTIN(0) haben. Ist dieser Wert negativ, wird eine Fläche gefüllt, deren jetzige Farbe ABS(INTIN(0)) ist. Diese Routine wertet außerdem die Farb- und Clippingvariablen aus. Das Füllmuster dieser Fläche wird wie beim $A005-Befehl festgelegt.

Line-F

Unter Line-F versteht man »Opcodes des MC68000 Prozessors, deren höchste vier Bits den Wert %1111=$F haben. Trifft der Prozessor auf einen dieser Opcodes, springt er über den Exceptionvektor Nummer 11 (steht in Adresse $00002C) in eine bestimmte Routine, die diesen Opcode bearbeitet. Normalerweise werden diese Opcodes zur Programmierung des Floating Point Coprozessors MC68881 benutzt. Dies ist im Atari ST nicht möglich. Man hatte bei Atari Probleme, das Betriebssystem in die *nur* 192 kByte fassenden PROMs zu bekommen. Der größte Teil dieses Betriebssystems ist in C geschrieben und jede Routine wird in C etwa so beendet:

```
movem.l  (SP)+,Dx-Dy/Ax-Ay
unlk     A6
rts
```

Diese Befehlssequenz holt zuerst die in der Routine benutzten Registerinhalte wieder vom Stack, setzt dann den Stackpointer wieder auf seinen alten Wert und kehrt zuletzt ins aufrufende Programm zurück. Insgesamt werden hierfür vier Worte benötigt. Statt diesen vier Worten kommt man nun durch Benutzung der Line-F Be-

fehle mit einem Wort aus. Dazu wird folgendermaßen vorgegangen. Das Wort hinter dem Opcode für **movem.l** gibt an, welche Register übertragen werden. Die Bits 0 bis 7 entsprechen den Datenregistern 0 bis 7 und die Bits bis 15 entsprechen den Adreßregistern 0 bis 7. Dieses Wort wird nun um zwei Bits nach rechts geschoben, Bit 0 und die Bits 12-15 werden gesetzt. Das resultierende Wort wird dann statt der obigen Befehlssequenz eingesetzt. Die Line-F Routine sieht nach, ob Bit 0 gesetzt ist. Wenn ja, werden die Bits 11-1 zum Zurückholen der entsprechenden Registerinhalte benutzt. Somit können die Register D3 bis D7 und A0 bis A5 vom Stack zurückgeholt werden.

Ist Bit 0 nicht gesetzt, dann darf auch Bit 1 nicht gesetzt sein, und die Bits 11-2 werden als Offset in eine Sprungtabelle benutzt. Diese Tabelle enthält somit 1024 Vektoren, deren Funktion noch nicht vollständig geklärt ist.

Maus
Zur Gattung der Säugetiere gehörendes Tier. In der Computerei bezeichnet man ein bestimmtes Gerät, welches Daten vom Benutzer zum Computer saugt, als M.

Aber Spaß beiseite: bei der M. handelt es sich um ein Eingabegerät, welches die Positionierung eines Zeigers (des »Mauszeigers) auf dem Bildschirm erlaubt, in dem man die M. auf dem Tisch oder einem speziellen Tablett verschiebt. Es gibt optische M., die eine spezielle Unterlage benötigen, und mechanische M. (wie die des Atari), die nur eine glatte Unterlage brauchen. Mit der M. begann der Siegeszug der grafischen Benutzeroberflächen bei Personal Computern, bei denen keine Kommandos mehr eingetippt zu werden brauchen, sondern der Benutzer auf die gewünschte (oft symbolisierte) Funktion zeigt und diese durch Drücken einer Taste (meist auf der Maus selbst befindlich) auswählt.

Untersuchungen haben gezeigt, daß beispielsweise die Cursorsteuerung mit der M. am einfachsten, praktischsten und schnellsten geht.

Mauscursor
siehe »Mauszeiger

Mausknopf
Wird auch als "Button" ("Mousebutton") bezeichnet. Wenn Sie sich die »Maus für den Atari betrachten, so werden Sie feststellen, daß sie an der Oberseite zwei Knöpfe hat. Dies sind die M.

Ob ein Benutzer einen dieser Knöpfe gedrückt hat, können Sie mittels »*vq_mouse()* oder »*graf_mkstate()* herausfinden.

Maustaste
Siehe »Mausknopf

Mauszeiger
Wenn die »Maus bewegt wird, so bewegt sich zugleich auf dem »Bildschirm der M. Dabei handelt es sich um einen »Sprite, der meistens die Form eines Pfeils hat (andere Formen sind möglich, siehe z. B. »*graf_mouse()*. Die Daten für den M. sind in einem 37 Worte umfassenden Feld gespeichert, welches folgenden Aufbau hat:

word 0	X-Koordinate des "Hot Spot"
word 1	X-Koordinate des "Hot Spot"
word 2	Reserviert, muß gleich eins sein
word 3	Farbe der Maske, normalerweise 0
word 4	Farbe der Daten, normalerweise 1
word 5 - 20	16 Wörter mit den Daten der Form
word 21 - 36	16 Wörter mit den Daten der Maske

Der M. hat eine Größe von 16 mal 16 Pixeln (bei jeder Auflösung).

Der "Hot Spot" bezeichnet den Punkt, der als X-Y-Koordinate bei der Abfrage der Mausposition zurückgegeben wird.

Die Maske wird benutzt, um auf schwarzen Flächen eine weiße Umrandung um den Mauszeiger zu erzeugen.

MC68000

Im Atari ST versieht ein Mikroprozessor vom Typ MC68000 seinen Dienst. Dieser Prozessor besitzt einen nicht gemultiplexten 16 Bit breiten Daten- und 24 Bit breiten Adreßbus. Damit läßt sich ein linearer Speicherbereich von 16 MByte adressieren. Der im Atari ST eingesetzte Memory Controller läßt aber nur 4 MByte zu, was aber für die meisten Anwendungen ausreichen sollte.

Diese CPU besitzt je 8 völlig gleichberechtigte Daten- und Adreßregister mit einer Breite von jeweils 32 Bit. Die Datenregister werden mit D0 bis D7 und die Adreßregister mit A0 bis A7 bezeichnet. Das Register A7 ist zweifach vorhanden (»Supervisormode) und wird im allgemeinen als Stackpointer benutzt. Das Statusregister hat eine Breite von 16 Bit und hat folgenden Aufbau:

Bit 0	C-Flag (Carry). Zeigt einen Übertrag bei Additionen oder Subtraktionen an.
Bit 1	V-Flag (Overflow). Zeigt Überläufe an.
Bit 2	Z-Flag (Zero). Ein gesetztes Bit bedeutet, daß das Resultat der letzten Operation Null war.
Bit 3	N-Flag (Negativ). Enthält das höchstwertigste Bit der letzten Arithmetikoperation.
Bit 4	X-Flag (Extend).
Bit 5-7	Diese Bits des Userstatusbytes werden nicht benutzt.
Bit 8-10	I0-I2. Diese Interruptmaske gibt den internen Interruptstatus der CPU an.
Bit 11-12	werden nicht benutzt.
Bit 13	S-Flag (»Supervisormode).
Bit 14	wird nicht benutzt.
Bit 15	T-Flag (»Tracebit).

Diese CPU versteht 56 verschiedene Befehle (»Opcodes) mit jeweils bis zu 13 Adressierungsarten, kann »Interrupts mit 7 verschiedenen Leveln bearbeiten und in zwei verscheidenen Modi arbeiten (»Supervisormode und »Usermode).

Media-Change

M.-C. heißt z.B. bei Disketten, daß eine Diskette gewechselt wurde. M.-C. wird vom Atari per Interrupt abgefragt, wobei ein Driveselect ausgeführt wird und getestet wird, ob die Diskette schreibgeschützt ist. Wenn sie schreibgeschützt ist, wird das als Diskettenwechsel aufgefaßt. Danach lädt der Rechner den Bootsektor und vergleicht die Seriennummer, wenn sie gleich ist wurde die Diskette nicht gewechselt, sonst aber wird auf M.-C. erkannt. Darin liegt eine Gefahr, wenn zwei Disketten, die gleiche Seriennummer besitzen wird nicht auf M.-C. erkannt.

Memory Form Definition Block

Bei dem M. (Abk.: MFDB) handelt es sich um eine Struktur, die rechteckige Bildausschnitte definiert. Diese bezeichnet man als »Raster. Der Bildausschnitt kann sich im Video-Speicher oder im restlichen Speicher befinden.

Die Struktur besitzt folgenden Aufbau:

Offs.	Typ	Name	Erläuterung
0	long	fd_addr	Die Adresse des Rasters
4	word	fd_w	Breite des Rasters in »Pixeln
6	word	fd_h	Höhe in »Pixeln
8	word	fd_wdwidth	Breite in Wörtern
10	word	fd_stand	Flag: zeigt an, ob das Raster in Standard- oder gerätespezifischer Form ist
12	word	fd_nplanes	Anzahl der Farbebenen (bzw. Anzahl der Bits pro »Pixel)
14	word	fd_r1	Reserviert
16	word	fd_r2	Reserviert
18	word	fd_r3	Reserviert

Mehr über diese Struktur können Sie unter dem Stichwort »Raster nachlesen.

Metafile

Ein M. ist eine Datei, in der die Informationen für ein Objekt-orientiertes Zeichenprogramm abgelegt werden. Objekt-orientiert bedeutet, daß die Daten nicht wie bei einem »Bit-Image organisiert sind (also pixelweise), sondern daß Zeichenobjekte be-

schrieben werden, z. B. ein »Generalized Drawing Primitive. Diese Art von Objekten hat **nichts** mit dem unter diesem Stichwort beschriebenen Gebilden zu tun! Wenn also etwa die Lage und Größe eines Rechtecks (dies wäre so ein Objekt) festgelegt wird, so steht in einem M. zunächst die Funktionsnummer der Routine, die dieses Objekt zeichnen kann (der Opcode), die Anzahl der Integer-Parameter, die Anzahl der Koordinatenpaare sowie die Parameter des Rechtecks selbst.

Bleiben wir bei diesem Beispiel und sehen wir uns den Aufbau noch etwas genauer an. Unter dem Stichwort »VDI haben wir Ihnen beschrieben, wie ein Funktionsaufruf von »VDI-Routinen aussieht. Im wesentlichen geschieht die Parameterübergabe durch drei Arrays, deren Namen **contrl**, **ptsin** und **intin** lauten (in den anderen Arrays bekommen Sie Werte zurück). Sehen Sie sich mal die Funktion »*v_bar()* an. Diese zeichnet ein Rechteck. Dort ist auch angegeben, mit welchen Werten die drei Arrays zu füllen sind. In **contrl(0)** steht der Opcode (= Funktionsnummer) der Funktion, in **contrl(1)** die Anzahl der Koordinatenpaare, die das Rechteck definieren (zwei sich diagonal gegenüberliegende Ecken des Rechtecks, dies sind die Werte für das **ptsin**-Array), in **contrl(3)** die Anzahl der Parameter im **intin**-Array und in **contrl(5)** ein Sub-Opcode.

Und in genau der beschriebenen Reihenfolge finden Sie die Werte in einem M. für unser Rechteck auch wieder! In diesem (standardisierten) Format legt beispielsweise ein Programm wie "GEM-DRAW" seine Daten ab ("GEM-DRAW" ist ein Warenzeichen der Firma Digital Research). Hinter den oben erklärten Werten liegen dann die eigentlichen Zeichen-Daten, deren Anzahl ja oben festgelegt wurde.

Somit hat unser Rechteck in einem M. folgendes Aussehen:

Wort-Nr.	Wert	Array	Beschreibung
0	11	**contrl(0)**	Opcode (Funktionsnummer), in unserem Beispiel ist das der von »*v_bar()*
1	2	**contrl(1)**	Anzahl der Werte im **ptsin**-Array, hier sind es zwei (zwei Koordinatenpaare)
2	0	**contrl(3)**	Anzahl der Werte im **intin**-Array (siehe »*v_bar()*)
3	1	**contrl(5)**	Sub-Opcode, in diesem Fall die Nummer des »GDP
4-7	?	**ptsin(0-3)**	Die Koordinaten zweier diagonal gegenüberliegender Ecken
-	-	**intin(0-)**	Hier folgten jetzt die dem Wert von **contrl(3)** entsprechende Anzahl von Integer-Werten; in unserem Beispiel also keine

Ein M. besteht also aus einer Folge von Parameterblöcken, die zum Aufruf der jeweils benötigten »VDI-Funktion gehören. Zusätzlich gibt es aber noch den sog. "Metafile-header". Dieser hat folgenden Aufbau:

Wort-Nr.	Bedeutung
0	$ffff = Dateianfang (derselbe Wert wird auch als Ende-Kennzeichnung verwendet)
1	Hauptversionsnummer
2	Nebenversionsnummer
3	0 = es werden »normalised Device Coordinates verwendet 2 = es werden »Raster-Koordinaten benutzt
4 - 7	Vier Werte, mit denen ein Rahmen beschrieben wird, welches die komplette Grafik umschließt: Wort(4) = kleinsmöglicher X-Wert Wort(5) = kleinsmöglicher Y-Wert Wort(6) = größtmöglicher X-Wert Wort(7) = größtmöglicher Y-Wert Wenn diese Werte nicht benötigt werden, so werden sie auf Null gesetzt.
8 - 15	reserviert (werden auf Null gesetzt)

Das Ende der Datei wird ebenfalls mit $ffff gekennzeichnet. Dies geschieht durch Aufruf von »*v_clswk()*. Normalerweise wird ein M. erzeugt, wenn mittels »GDOS ein Metafile-Treiber geladen wurde (META.SYS) und eine »Workstation geöffnet wird. Aber: die Funktion »*v_opnwk()* zum Öffnen einer »Workstation funktioniert nicht (ebensowenig wie die Routine »*v_clswk()* zum Schließen einer »Workstation). Daher muß die Arbeit, einen Metafile-Header zu erzeugen, vom Programm selbst getätigt werden. Auch das Ende des Metafiles muß vom Programm aus mit $ffff gekennzeichnet werden.

Folgende Routinen sind Standard-Metafile-Items:

Funktionsnr.	Funktionsnamen
$03 = 3	v_clrwk(handle)
$04 = 4	v_updwk(handle)
$05 = 5	v_clear_disp_list(handle)
$05 = 5	v_output_window(handle, xyarray)
$05 = 5	v_enter_cur(handle)
$05 = 5	v_exit_cur(handle)
$05 = 5	v_form_adv(handle)
$06 = 6	v_pline(handle, count, pxyarray)
$07 = 7	v_pmarker(handle, count, pxyarray)
$08 = 8	v_gtext(handle, x, y, string)
$09 = 9	v_fillarea(handle, count, pxyarray)
$0B = 11	v_bar(handle, pxyarray)
$0B = 11	v_pieslice(handle, x, y, radius, begang, endang)

Funktionsnr.	Funktionsnamen
$0B = 11	v_justified(handle, x, y, string, length, word_space, char_space)
$0B = 11	v_ellarc(handle, x, y, xradius, yradius, begang, endang)
$0B = 11	v_rfbox(handle, xyarray)
$0B = 11	v_rbox(handle, xyarray)
$0B = 11	v_circle(handle, x, y, radius)
$0B = 11	v_ellipse(handle, x, y, xradius, yradius)
$0B = 11	v_ellpie(handle, x, y, xradius, yradius, begang, endang)
$0B = 11	v_arc(handle, x, y, radius, begang, endang)
$0C = 12	vst_height(handle, height, &char_width, &char_height &cell_width, &cell_height)
$0D = 13	set_baseline = vst_rotation(handle, angle)
$0E = 14	vs_color(handle, index, rgb_in)
$0F = 15	set_type = vsl_type(handle, style)
$10 = 16	set_width = vsl_width(handle, width)
$11 = 17	set_color = vsl_color(handle, color_index)
$12 = 18	set_type = vsm_type(handle, symbol)
$13 = 19	set_height = vsm_height(handle, height)
$14 = 20	set_color = vsm_color(handle, color_index)
$15 = 21	set_font = vst_font(handle, font)
$16 = 22	set_color = vst_color(handle, color_index)
$17 = 23	set_interior = vsf_interior(handle, style)
$18 = 24	set_style = vsf_style(handle, style_index)
$19 = 25	set_color = vsf_color(handle, color_index)
$20 = 32	set_mode = vswr_mode(handle, mode)
$27 = 39	vst_alignment(handle, hor_in, vert_in, &hor_out, &vert_out)
$68 = 104	set_perimeter = vsf_perimeter(handle, per_vis)
$6A = 106	set_effects = vst_effects(handle, effect)
$6B = 107	set_point = vst_point(handle, point, &char_width, &char_height &cell_width, &cell_height)
$6C = 108	vsl_ends(handle, beg_style, end_style)
$70 = 112	vsf_udpat(handle, pfill_pat, planes)
$71 = 113	vsl_udsty(handle, pattern)
$72 = 114	vr_recfl(handle, pxyarray)
$81 = 129	vs_clip(handle, clip_flag, pxyarray)

Des weiteren gibt es noch drei weitere »VDI-Routinen zur Behandlung von M. Diese sind:

Funktionsnr.	Funktionsnamen
$05 = 5	vm_filename(handle, filename)
$05 = 5	v_meta_extents(handle, min_x, min_y, max_x, max_y)
$05 = 5	v_write_meta(handle, num_intin, intin, num_ptsin, ptsin)

MFDB

Siehe »Memory Form Definition Block

MFP 68901

Abk.: Multi Function Peripheral. Dieser Chip bietet eine wirklich hohe Funktionalität. Er besitzt einen 8 Bit breiten Parallelport, dessen Pins alle separat als Ein- oder Ausgang programmierbar sind. Diese Portbits können als Interruptquellen benutzt werden. Der Chip besitzt eine serielle Schnittstelle, die jedoch leider nicht vollständig ist, da die Statusleitungen für eine »V.24-Schnittstelle fehlen. Dafür enthält das IC vier Timer und bietet insgesamt 16 verschiedene Interruptmöglichkeiten an. Die 24 Register dieses Chips sind an den ungeraden (!) Adressen von $FFFA01 (Register 0) bis $FFFA2F (Register 23) erreichbar.

Kommen wir nun zur Beschreibung dieser Register und ihrer Funktion im Atari ST. Die Registernamen entstammen der Mostek-Beschreibung. Dieses IC kümmert sich fast um die gesamte Interruptsteuerung im Atari ST. Es kann in zwei verschiedenen Modi arbeiten, die durch Bit 3 in Register 11 unterschieden werden. Ist dieses Bit gesetzt, was im Atari ST normalerweise der Fall ist, dann arbeitet der Chip im *Software End of Interrupt* Modus (SEI), ansonsten im *Automatic End of Interrupt* Modus (AEI). Löst eine der Quellen einen Interrupt aus, sieht das Mikroprogramm des Chips zunächst im **IER** nach, ob der Interrupt erlaubt ist. Ist das zugehörige Bit gelöscht, wird der Interrupt gar nicht erst weiter bearbeitet. Ansonsten wird der CPU ein Interrupt gemeldet und die entsprechenden Bits in den Registern **IPR** und **ISR** werden gesetzt. Wird dann der Interrupt von der CPU bearbeitet, legt der Chip einen dem Interrupt entsprechenden Vektor auf den Datenbus und löscht das Bit im **IPR**. Dieser Vektor stellt eigentlich nur eine Vektornummer dar, die einen Wert von $40-$FF haben kann. Sie wird im MFP 68901 folgendermaßen gebildet. Die Bits 0-3 entsprechen der Interruptquelle und die Bits 4-7 werden den Bits 4-7 des Registers 11 entnommen. Diese Bits sollten den Wert %0100 haben. Die CPU multipliziert diese Vektornummer mit 4 und erhält dadurch die Adresse, in der die eigentliche Startadresse der Interrupt-Service-Routine steht ($000100-$0003FF »Interruptstruktur). Arbeitet der Chip nun im AEI-Modus, dann wird das entsprechende Bit im **ISR** sofort nach dem Senden der Vektornummer gelöscht. Im SEI-Modus muß das Löschen von der Interrupt-Service-Routine übernommen werden. Löst nämlich dieselbe Quelle einen weiteren Interrupt aus, während der erste noch in Arbeit ist (Bit im **ISR** ist gesetzt), so wird er erst dann bearbeitet, wenn das Bit im **ISR** gelöscht wurde.

REG 0 = **GPIP, General Purpose I/O Port.** Die Bits dieses Registers entsprechen den Zuständen der Pins am Parallelport. Das Register kann sowohl gelesen als auch beschrieben werden.

REG 1 = **AER, Active Edge Register.** Die Portpins können als Interruptquellen benutzt werden. Die Bits dieses Registers legen dann fest, ob ein Interrupt bei einem Low/High-Übergang (Bit gesetzt) oder bei einem High/Low-Übergang (Bit gelöscht) ausgelöst werden soll.

REG 2 = **DDR, Data Direction Register.** Diese Bits legen fest, ob der zugehörige Portpin als Eingang (Bit gelöscht) oder als Ausgang (Bit gesetzt) arbeiten soll.

REG 3 = **IERA, Interrupt Enable Register A.** Die Bits dieses und des nächsten Registers erlauben einer Interruptquelle, einen Interrupt auszulösen, wenn das entsprechende Bit gesetzt ist. Es gilt folgende Zuordnung der Bits zu den Interruptquellen:

Bit 7 = I/O-Port Bit 7 (höchste Priorität)
Bit 6 = I/O-Port Bit 6
Bit 5 = Timer A abgelaufen
Bit 4 = Empfangspuffer ist voll
Bit 3 = Fehler beim Empfang aufgetreten
Bit 2 = Sendepuffer ist leer
Bit 1 = Fehler beim Senden aufgetreten
Bit 0 = Timer B abgelaufen

REG 4 = **IERB, Interrupt Enable Register B.**

Bit 7 = I/O-Port Bit 5
Bit 6 = I/O-Port Bit 4
Bit 5 = Timer C
Bit 4 = Timer D
Bit 3 = I/O-Port Bit 3
Bit 2 = I/O-Port Bit 2
Bit 1 = I/O-Port Bit 1
Bit 0 = I/O-Port Bit 0

Diese Zuordnung gilt ebenfalls für alle folgenden Doppelregister.

REG 5+6 **IPRA+IPRB, Interrupt Pending Register.** Wenn eine der Interruptquellen einen Interrupt ausgelöst hat, wird hier im IPR das zugehörige Bit gesetzt. Es wird automatisch gelöscht, wenn der Chip bei der Vektorerzeugung die zugehörige Vektornummer auf den Datenbus gelegt hat.

REG 7+8 **ISRA+ISRB, Interrupt in-Service Register.** Ein gesetztes Bit im ISR zeigt an, daß die zugehörige Interrupt-Routine noch nicht beendet wurde. Dadurch werden alle nachfolgenden Interrupts mit kleinerer oder gleicher Priorität nicht bearbeitet.

REG 9+10 **IMRA+IMRB, Interrupt Mask Register.** Ein gesetztes Bit verhindert, daß der zugehörige Interrupt an die CPU gemeldet wird, obwohl er durch das IER erlaubt wurde. Intern wird jedoch völlig gleich verfahren. Diese

Möglichkeit wird im Atari ST nicht benutzt und dient dazu, diesen Chip auch mit anderen CPU's zusammenarbeiten zu lassen.

REG 11 **VR, Vector Register.** Bit 0-2 dieses Registers werden nicht benutzt. Ist Bit 3 gesetzt, arbeitet der Chip im *Software End of Interrupt*, sonst im *Automatic End of Interrupt* Modus (siehe oben).

REG 12+13 **TACR+TBCR, Timer A/B Control Register.** Die Bits 3-0 dieser Register haben folgende Bedeutung:

3210	Bedeutung
%0000	- Timer Reset. Der Timer hat keine Funktion.
%0001	- Delay Mode. Vorteiler teilt durch 4.
%0010	- Delay Mode. Vorteiler teilt durch 10.
%0011	- Delay Mode. Vorteiler teilt durch 16.
%0100	- Delay Mode. Vorteiler teilt durch 50.
%0101	- Delay Mode. Vorteiler teilt durch 64.
%0110	- Delay Mode. Vorteiler teilt durch 100.
%0111	- Delay Mode. Vorteiler teilt durch 200.
%1000	- Event Count Mode. Es werden die Impulse an den Pins TAI (19) bzw. TBI (20) gezählt.
%1001	- Pulsweitenmessung. Vorteiler teilt durch 4.
%1010	- Pulsweitenmessung. Vorteiler teilt durch 10.
%1011	- Pulsweitenmessung. Vorteiler teilt durch 16.
%1100	- Pulsweitenmessung. Vorteiler teilt durch 50.
%1101	- Pulsweitenmessung. Vorteiler teilt durch 64.
%1110	- Pulsweitenmessung. Vorteiler teilt durch 100.
%1111	- Pulsweitenmessung. Vorteiler teilt durch 200.

Wird Bit 4 dieses Registers gesetzt, dann werden die Timerausgänge (TAO=Pin 13 bzw. TBO=Pin 14) Low.

REG 14 **TCDCR, Timer C and D Control Register.** Die beiden Timer C und D arbeiten nur im Delay-Modus, so daß je drei Bit zur Steuerung ausreichen. Die Bits 3 und 7 haben keine Funktion. Die Bits 6-4 gelten für Timer C und die Bits 2-0 für Timer D:

654 210	Bedeutung
%000	- Timer Reset.
%001	- Delay-Mode. Vorteiler teilt durch 4.
%010	- Delay-Mode. Vorteiler teilt durch 10.
%011	- Delay-Mode. Vorteiler teilt durch 16.
%100	- Delay-Mode. Vorteiler teilt durch 50.
%101	- Delay-Mode. Vorteiler teilt durch 64.
%110	- Delay-Mode. Vorteiler teilt durch 100.
%111	- Delay-Mode. Vorteiler teilt durch 200.

REG 15-18 **TADR+TBDR+TCDR+TDDR, Timer x Data Register.** Jedes dieser 8 Bit breiten Register nimmt einen Wert auf. Dieser wird bei jedem Impuls dekrementiert. Beim

Erreichen des Wertes Null wird ein Interrupt angezeigt und der Timer wieder neu mit diesem Wert geladen.

REG 19 **SCR, Synchronous Character Register.** Arbeitet die serielle Schnittstelle dieses Chips im synchronen Modus, dann wird beim Empfang jedes Zeichen mit dem Inhalt dieses Registers verglichen. Erst nach Übereinstimmung werden Zeichen in den Empfangspuffer übernommen.

REG 20 **UCR, USART Control Register.** Dieses Register steuert den *Universal Synchronous/Asynchronous Receiver/Transmitter*. Die einzelnen Bits haben folgende Bedeutung:

Bit 0 = Nicht benutzt.

Bit 1 = Parität gerade (1), oder ungerade (0).

Bit 2 = Parität benutzt (1), oder unbenutzt (0).

Bit 3+4

43	Startbits	Stopbits	Format
%00	0	0	Synchron
%01	1	1	Asynchron
%10	1	1.5	Asynchron
%11	1	2	

Bit 5+6 Bestimmen die Anzahl der Datenbits.

65	Datenbits
%00	8
%01	7
%10	6
%11	5

Bit 7 = Ist dieses Bit gesetzt, wird die Sende- und Empfangsfrequenz intern durch 16 geteilt. Ist es gelöscht, bleibt sie unverändert.

REG 21 **RSR, Receiver Status Register.** Die Bits dieses Registers geben Auskunft über den Zustand des Empfängers:

Bit 0 = Einschalten des Empfängers (Bit = 1).

Bit 1 = Gesetztes Bit sorgt dafür, daß ein mit dem Wert im SCR übereinstimmendes Zeichen zusätzlich in den Empfangspuffer übernommen wird.

Bit 2 = Ein gesetztes Bit zeigt im synchronen Modus an, daß ein mit dem Wert im SCR übereinstimmendes Zeichen gefunden wurde. Im asynchronen Modus wird dieses Bit nach einem Startbit gesetzt und nach dem Stopbit wieder gelöscht.

Bit 3 = Im synchronen Betrieb zeigt dieses Bit an, daß das empfangene Zeichen mit dem Wert im SCR übereinstimmt. Im asynchronen Betrieb zeigt es den Empfang eines Break-Zeichens an.

Bit 4 = Zeigt einen **Frame-Error** an, d.h. es wurde ein Zeichen empfangen, dessen Stopbit Null ist.

Bit 5 = **Parity-Error.** Bei eingeschalteter Paritätsprüfung zeigt dieses Bit ein Zeichen mit Paritätsfehler an.

Bit 6 = **Overrun-Error.** Dieser Fehler tritt auf, wenn das zuletzt empfangene Zeichen noch nocht abgeholt wurde und das nächste Zeichen bereits komplett empfangen worden ist.

Bit 7 = Dieses Bit zeigt an, daß sich im Empfangspuffer ein Zeichen befindet, das darauf wartet, vom Prozessor ausgelesen zu werden. Ein Lesezugriff auf das Datenregister löscht dieses Bit wieder.

REG 22 **TSR, Transmitter Status Register.** Enthält den Status des Transmitters:

Bit 0 = Ist dieses Bit gesetzt, arbeitet der Sendeteil des Chips normal, ansonsten ist er ausgeschaltet.

Bit 1+2 Diese beiden Bits legen den Zustand des TX-Pins bei ausgeschaltetem Transmitter fest.

21	Bedeutung
%00	TX-Pin ist im Tri-State Zustand
%01	Der TX-Pin ist konstant Low.
%10	Der TX-Pin ist konstant High.
%11	TX-Pin ist high und Loopback-Modus ist an.

Bit 3 = Im asynchronen Modus wird der MFP 68901 durch Setzen dieses Bits dazu gebracht, ein Break-Zeichen zu senden.

Bit 4 = Dieses Statusbit zeigt an, daß der Transmitter ausgeschaltet ist.

Bit 5 = **Auto-Turnaround.** Ist dieses Bit gesetzt, wird im asynchronen Modus der Empfänger automatisch eingeschaltet, wenn der Transmitter ausgeschaltet wird und das letzte zu sendende Zeichen vollständig gesendet wurde.

Bit 6 = **Underrun-Error.** Zeigt an, daß das letzte Zeichen vollständig gesendet wurde und der Prozessor noch nicht für Nachschub gesorgt hat.

Bit 7 = **Buffer empty.** Zeigt an, daß der Sendepuffer bereit ist, das nächste Zeichen entgegenzunehmen.

REG 23 **UDR, USART Date Register.** Erfolgt ein Schreibzugriff auf dieses Register, so wird das Zeichen in den Sendepuffer übertragen. Bei einem Lesezugriff wird ein Zeichen aus dem Empfangspuffer gelesen.

Menu_bar

Zur »Menu Library gehörende Routine.

```
$1E = 30        mebreturn = menu_bar(me_btree, me_bshow)
word            me_breturn,                    me_bshow;
word *                          me_btree;
```

Mit dieser Routine können Sie die Menüleiste am oberen Rand des Bildschirms für die »Drop-Down-Menüs darstellen. Sobald sich Ihr

Programm in der »*evnt_multi()*-Schleife befindet (oder mit »*evnt_-mesag()* auf eine »Nachricht wartet), schickt »AES dann eine »Nachricht, sobald der Benutzer einen Menüpunkt ausgewählt hat. Für die Überwachung der Menüs ist alleinig »AES verantwortlich (genauer: der »Screen Manager).

Mit Beendigung des Programms sollte diese Routine dazu verwendet werden, um die Menüs wieder abzuschalten.

Übrigens: Falls Ihre Menüs vom Mauspfeil zerstört werden, liegt das daran, daß sich die Befehle zum Mauspfeil Zeigen und Verstekken nicht die Balance halten (siehe »*graf_mouse()*). Es ist (mindestens) ein Aufruf zum Vorzeigen der Maus zuviel vorhanden!

Eingabeparameter:

word *me_btree — Die Adresse des »Objektbaums, der die Menüstruktur enthält. Dieser wird (meist) mithilfe eines »Resource Construction Sets erstellt und dann die zugehörige Adresse mit »*rsrc_gaddr()* bestimmt.

word me_bshow — Dieses Flag bestimmt, ob der Baum dargestellt ('me_bshow' = 1) oder gelöscht ('me_bshow' = 0) wird.

```
contrl(0)   =  30
contrl(1)   =   1
contrl(3)   =   1

int_in(0)   =  me_bshow

addr_in(0)  =  me_btree
```

Ausgabeparameter:

word me_breturn — 0 = Fehler, n = alles Ok.

```
contrl(2)   =   1
contrl(4)   =   0

int_out(0)  =  me_breturn
```

Menu_icheck

Zur »Menu Library gehörende Routine.

```
$1F = 31        me_creturn = menu_icheck(me_ctree, me_citem, me_ccheck)
word            me_creturn,                          me_citem, me_ccheck;
word *                                      me_ctree:
```

Manchmal ist es sinnvoll, in einem »Drop-Down-Menü anzuzeigen, ob eine bestimmte Funktion in Kraft ist. Dazu kann man vor dem Menütext (vorausgesetzt, man hat genug Platz vor dem Text gelassen) ein Häkchen darstellen. Diese Routine ist dafür zuständig. Nach allgemeiner Konvention läßt man vor dem Menü-Text stets zwei Leerzeichen.

Eingabeparameter:

word *me_ctree	Die Adresse des »Objektbaums, in dem die Datenstruktur für das Menü ist.
word me_citem	Die Indexnummer des Menüpunktes in dem »Objektbaum. Dieser Nummer kann man im »Resource Construction Programm einen symbolischen Namen zuweisen.
word me_ccheck	Wenn 'me_ccheck' = 1, dann wird links vor dem Text eine Markierung angebracht (ein Häkchen), wenn 'me_ccheck' = 0 ist, wird die Markierung gelöscht (falls vorhanden).

```
contrl(0)   =  31
contrl(1)   =   2
contrl(3)   =   1

int_in(0)   =  me_citem
int_in(1)   =  me_ccheck

addr_in(0)  =  me_ctree
```

Ausgabeparameter:

word me_creturn	0 = Ein Fehler ist vorgekommen, n = alles in Ordnung.

```
contrl(2)   =   1
contrl(4)   =   0

int_out(0)  =  me_creturn
```

Menu_ienable

Zur »Menu Library gehörende Routine.

```
$20 = 32        me_ereturn = menu_ienable(me_etree, me_eitem,
word            me_ereturn,                          me_eitem;
word *                                  me_etree;
                                        me_eenable)
word                                    me_eenable;
```

Mit dieser Routine ist es möglich, einen Menüpunkt aktivierbar zu machen, oder einen aktivierbaren Menüeintrag zu deaktivieren. Einen deaktivierten Menüpunkt erkennt man daran, daß die Schrift hell ist (Schattenschrift) und der Menütext nicht invertiert wird, wenn der Mauszeiger ihn berührt. Einen solchen Menüpunkt kann der Benutzer nicht auswählen. Ein aktivierbarer Menüpunkt ist in normaler Schrift dargestellt.

Die beste Methode, Fehleingaben des Benutzers abzufangen, besteht darin, die Eingabe erst garnicht zuzulassen. Daher ist es sinnvoll, unmögliche Menüpunkte abzuschalten.

Eingabeparameter:

word *me_etree	Die Adresse des »Objektbaumes, der die Menüstruktur enthält (siehe auch »*menu_bar()*).
word me_eitem	Die Indexnummer des Menüpunktes (siehe auch die Erläuterung der Variablen 'me_ccheck' in der Routine »*menu_icheck()*).
word me_eenable	Ein Flag, das anzeigt, wie der Menüpunkt dargestellt werden soll. Wenn me_eenable = 1, dann wir der Menüpunkt in normaler Schrift dargestellt und ist für den Benutzer verfügbar, wenn me_eenable = 0, so wird der Menüpunkt in heller Schrift dargestellt und ist für den Benutzer nicht wählbar.

```
contrl(0)   =  32
contrl(1)   =   2
contrl(3)   =   1

int_in(0)   =  me_eitem
int_in(1)   =  me_eenable

addr_in(0)  =  me_etree
```

Ausgabeparameter:

word me_ereturn 0 = Ein Fehler ist vorgekommen, n = alles Ok.

contrl(2) = 1
contrl(4) = 0

int_out(0) = me_ereturn

Menü

Eine weitverbreitete Methode, mehrere Möglichkeiten zur Auswahl anzubieten (z. B. Speisekarte). Beim Computer bezeichnet ein M. ebenfalls eine bestimmte Methode, um die Wahl verschiedener Funktionen zu erleichtern.

Menu Library

Diese Unterroutinensammlung dient dem Darstellen und Verändern von »Drop-Down-Menüs. Die M. besteht aus folgenden Routinen:

-»*menu_bar()*: Dient dazu, eine Menüleiste darzustellen oder wieder zu löschen.

-»*menu_icheck()*: Ermöglicht es, ein Häkchen ("Checkmark") vor einem Menüpunkt anzubringen oder zu entfernen.

-»*menu_ienable()*: Damit können Menüpunkte angeschaltet oder abgeschaltet werden. Aktivierbare Menüpunkte werden in normaler Schrift dargestellt, nicht aktivierbare in Schattenschrift (heller Schrift).

-»*menu_register()*: Mit dieser Routine können »Desk-Accessories einen String in das linke »Drop-Down-Menü schreiben und eine Identifikationsnummer erhalten (nicht zu verwechseln mit der »Task--ID).

-»*menu_text()*: Hiermit kann der Text eines Menüpunktes verändert werden.

-»*menu_tnormal()*: Der »Screen-Manager invertiert den Text der Menüleiste, wenn ein Menüpunkt gewählt wurde. Mit dieser Routine läßt sich das rückgängig machen (oder ein Menütitel invertieren).

Menu_register

Zur »Menu Library gehörende Routine.

```
$23 = 35        me_rmenuid = menu_register(me_rapid, me_rpstring)
word            me_rmenuid,                me_rapid;
byte *                                                 me_rpstring;
```

Diese Routine wird von »Desk-Accessories benötigt, um Menüpunkte in das linke »Drop-Down-Menü einzutragen Desk-Menü. Damit meldet ein »Desk-Accessory sein Vorhandensein an und macht sich dem Benutzer verfügbar, solange das Desk-Menü vom Programm angezeigt wird. Insgesamt können sich sechs »Desk-Accessories anmelden.

Eingabeparameter:

word me_rapid	Dies ist die »Task-ID des »Desk-Accessories, die für das Programm von der Routine »*appl_init()* zurückgegeben wird.
byte *me_rpstring	Die Adresse des Textes, unter dem das »Desk--Accessory in dem Desk-Menü erscheinen soll.

```
contrl(0)   =  35
contrl(1)   =   1
contrl(3)   =   1

int_in(0)   =  me_rapid

addr_in(0)  =  me_rpstring
```

Ausgabeparameter:

word me_rmenuid	Dieser Wert identifiziert das »Accessory bei weiteren Aufrufen. Wählt der Benutzer ein »Accessory, so sendet »AES die »Nachricht 'AC_-OPEN' (mehr über Nachrichten unter »Message).

```
contrl(2)   =   1
contrl(4)   =   0

int_out(0)  =  me_rmenuid
```

Menu_text

Zur »Menu Library gehörende Routine.

```
$22 = 34        me_treturn = menu_text(me_ttree, me_titem, me_ttext)
word            me_treturn,                    me_titem;
word *                                 me_ttree;
byte *                                                   me_ttext;
```

Mit dieser Routine können Sie den Text eines »Drop-Down-Menüs ändern. Damit ist es möglich, Menüeinträge je nach Kontext zu wechseln. Wenn ein Menü beispielsweise heißt "Einrücken ein", und der Benutzer aktiviert es (um das Einrücken von Text in der Textverarbeitung einzuschalten), so kann man den Text ändern auf "Einrücken aus". Beim nächsten aktivieren des Menüs wird das Einrükken dann wieder ausgeschaltet und der Text wieder auf "Einrücken ein" umgeschaltet usw.

Wichtig dabei ist, daß der neue Menüeintrag exakt dieselbe Länge haben **muß**, wie der alte. Ist er kürzer, so gibt es merkwürdige Effekte, ist er länger, wird die gesamte Menüstruktur zerstört (und das Programm "bombt"). Und noch etwas ist wichtig: der neue Menütext muß eine feste Adresse haben, da »AES nur einen Zeiger auf den Text speichert, nicht den Text selbst. Wird der Text verschoben oder gelöscht, so kann ein Benutzen des »Drop-Down-Menüs einen Systemabsturz verursachen.

Eingabeparameter:

word me_titem	Die Identifikationsnummer des Menüeintrags. Dies ist die Indexnummer des Menüpunktes im »Objektbaum.
word *me_ttree	Die Adresse des Menübaums (siehe »*menu_bar()*)
byte *me_ttext	Zeiger auf den Textstring, der neu eingetragen werden soll.

```
contrl(0)   =  34
contrl(1)   =   1
contrl(3)   =   2

int_in(0)   =  me_titem

addr_in(0)  =  me_ttree
addr_in(1)  =  me_ttext
```

Ausgabeparameter:

word me_treturn 0 = Ein Fehler ist aufgetreten, n = alles Ok.

contrl(2) = 1
contrl(4) = 0

int_out(0) = me_treturn

Menu_tnormal

Zur »Menu Library gehörende Routine.

```
$21 = 33        me_nreturn = menu_tnormal(me_ntree, me_ntitle, me_nnormal)
word            me_nreturn,                         me_ntitle, me_nnormal;
word *                                   me_ntree;
```

Damit kann man den Menütitel in der Menüleiste normal oder invertiert darstellen. Wenn der Mauszeiger den Menütitel berührt, so sorgt der »Screen Manager für eine Invertierung der Schrift (weiß auf schwarz). Verschwindet das Menü, ohne daß gewählt wurde, so wird die Schrift vom »Screen Manager wieder auf schwarz auf weiß geändert. Wird aber ein Menüeintrag gewählt, so ist die »Application selbst dafür verantwortlich, daß die Schrift wieder normal (schwarz auf weiß) dargestellt wird.

Angenommen, »AES schickt die »Message 'MN_SELECTED' (siehe »Message). So sorgt folgender Routinenaufruf wieder für eine normale Schrift:

```
   menu_tnormal(menu_lst, Msg[3], 1);
word *          menu_lst;
word                      Msg[3], 1;
```

wobei 'menu_lst' die Adresse der Menüstrukur ist (siehe »*menu_bar()*, Msg[] das Array, in das »AES seine Nachrichten schreibt (angegeben bei »*evnt_mesag()* und »*evnt_multi()*) und '1' ein Flag (s. u.).

Eingabeparameter:

word *me_ntree	Adresse der Menüstruktur
word me_ntitle	Die Indexnummer des Menütitels
word me_nnormal	Wenn 'me_nnormal' = 1 wird der Menütitel in normaler Schrift dargestellt, wenn 'me_nnormal' = 0 erfolgt die Darstellung in invertierter Schrift.

```
contrl(0)   =  33
contrl(1)   =   2
contrl(3)   =   1

int_in(0)   =  me_ntitel
int_in(1)   =  me_nnormal

addr_in(0)  =  me_ntree
```

Ausgabeparameter:

```
word me_nreturn     0 = Ein Fehler ist passiert, n = alles Ok.

contrl(2)   =  1
contrl(4)   =  0

int_out(0)  =  me_nreturn
```

Message

Zum »AES gehört der sog. »Screen-Manager, der alle Bewegungen der Maus verfolgt. Wenn der Mauszeiger beispielsweise den oberen Bildschirmrand erreicht - und damit ein Drop-Down-Menü - so übernimmt der »Screen-Manager die Kontrolle, um das entsprechende »Drop-Down-Menü zu zeichnen. Der »Screen-Manager gibt die Kontrolle wieder ab, wenn der Benutzer die linke Maustaste drückt und wieder losläßt. Das »Menü wird wieder gelöscht. Befand sich der »Mauszeiger beim Drücken der Maustaste über einem wählbaren Menüpunkt, so sendet »AES an alle Applikationen, die auf eine M. warten, eine entsprechende M. Jede Application kann dann testen, ob sie gemeint war und die gewünschten Aktionen durchführen.

Ähnliches geschieht, wenn eine Application eine Nachricht mittels *appl_write()* verschickt. Im Gegensatz zu den Mitteilungen von »AES sind diese aber nicht vordefiniert.

Die vordefinierten Mitteilungen:

Wenn auf eine Nachricht von »AES gewartet wird, so muß ein 16 Byte (8 Worte) großer Puffer angegeben werden, in den »AES die Nachricht schreibt. Die Worte haben dabei stets folgende Bedeutung:

Wort 0: Typ des Ereignisses
Wort 1: Die »Task-ID der Application, von der die Mitteilung kommt

Wort 2: Gibt die Länge der Mitteilung an, die über die 16 Byte hinausgeht. Ist diese größer als 16 Byte, so handelt es sich um eine mit »*appl_write()* geschriebene Nachricht, die mittels »*appl_read()* gelesen werden muß. Sonst findet man die Mitteilung in dem angegebenen Puffer.

Wort 3-7: Diverse Angaben, deren Bedeutung von der Art der Mitteilung abhängt, im folgenden beschrieben.

Die in der Beschreibung verwendeten Begriffe stehen für symbolische Zahlenwerte, die in der Datei "»GEMBIND.H" definiert worden sind (z. B. #define MN_SELECTED 10). Im weiteren werden stets diese symbolischen Namen benutzt, die zugehörige Nummer steht bei Wort 0:

Nachricht für »Drop-Down-Menüs

MN_SELECTED:

Steht im Puffer dieses Wort (= 10) als erstes, so hat der Benutzer eines der Drop-Down-Menüs ausgewählt.

word 0: 10

word 3: Die Nummer der gewählten Menüleiste (= die Nummer des »Objekts in der Titelleiste, die mit dem Resource Construction Program vergeben wurde).

word 4: Der im »Drop-Down-Menü gewählte Menüpunkt. Die entsprechende Nummer wird ebenfalls im »Resource Construction Set vergeben.

Nachrichten zur Fensterbehandlung:

Das im folgenden als "Nummer des Fensters" bezeichnete Wort ist der von »*wind_create()* (siehe dort) zurückgegebene Wert (=»Window-Handle), der zur weiteren Identifikation des Fensters dient.

Höhe und Breite werden stets in Pixeln angegeben - je nachdem, wie die entsprechende virtuelle Arbeitsstation geöffnet wurde, mit Raster- oder normalisierten Koordinaten.

Für ein Fenster, das keinen »Rollbalken besitzt, kann die Meldung "Rollbalken verschoben" (WM_HSLID oder WM_VSLID, siehe unten) niemals generiert werden - entsprechendes gilt für die anderen Fenster-Eigenschaften auch.

WM_REDRAW:

Zeigt an, daß ein Fenster (bzw. der Teil innerhalb eines Fensters) neu gezeichnet werden muß (wobei der Rahmen, die Titelzei-

le, die »Rollbalken und die Infozeile des Fensters stets von »AES neu gezeichnet werden). Wie das Neuzeichnen zu geschehen hat, ist bei einem eigenen Stichwort erklärt: »Neuzeichnen von Fenstern.

word 0: 20
word 3: Die Nummer des Fensters
word 4: X-Koordinate des neu zu zeichnenden Bereichs
word 5: Y-Koordinate des Bereichs
word 6: Die Breite des Bereichs in Pixel
word 7: Die Höhe des Bereichs in Pixel

WM_TOPPED:

Dies signalisiert, daß das entsprechende Fenster aktiviert werden soll - d. h., dieses Fenster soll zum obersten Fenster gemacht werden. Dies geschieht dann mittels eines »*wind_set()* Aufrufs. Nach dem Aufruf von »*wind_set()* generiert »AES eine WM_REDRAW-Nachricht (falls das Fenster teilweise von anderen Fenstern überlagert wurde). Dann, wenn das Fenster aktiviert worden ist, erzeugt »AES die Mitteilung "WM_NEWTOP" (siehe unten).

word 0: 21
word 3: Die Nummer des Fensters

WM_CLOSED:

Der Benutzer hat das Schließfeld (links oben) des Fensters angeklickt. Im Allgemeinen bedeutet das, daß das Fenster geschlossen werden soll. Selbstverständlich ist es aber möglich, daraufhin etwas anderes zu tun, z. B. anzunehmen, daß der Benutzer das Programm beenden möchte oder ähnliches.

word 0: 22
word 3: Die Nummer des Fensters

WM_FULLED:

Die Vergrößerungsbox im Fenster rechts oben wurde angeklickt. Im allgemeinen bedeutet dies, daß der Benutzer das Fenster auf seine volle Größe vergrössern möchte - oder, falls das Fenster bereits in voller Größe dargestellt ist, daß das Fenster auf seine vorige Größe zurückschrumpfen soll.

word 0: 23
word 3: Die Nummer des Fensters

WM_ARROWED:

Einer der »Rollbalken oder der »Rollpfeile ist angeklickt worden. Ein Anklicken der »Rollpfeile bedeutet, daß der Fensterinhalt um eine Zeile nach oben oder unten oder um eine Spalte nach links oder rechts gescrollt werden soll. Ein Anklicken der »Rollbalken bedeutet, daß das Fenster um eine Seite nach links, rechts, oben oder unten gescrollt werden soll. Was eine Seite, Zeile oder Spalte ist, wird dabei von der Art der im Fenster dargestellten Daten definiert.

```
word 0:     24
word 3:     Die Nummer des Fensters
word 4:     Hier steht, was genau angeklickt wurde:
            0 = Seite nach oben     (WA_UPPAGE aus GEMBIND.H)
            1 = Seite nach unten    (WA_DNPAGE)
            2 = Zeile nach oben     (WA_UPLINE)
            3 = Zeile nach unten    (WA_DNLINE)
            4 = Seite nach links    (WA_LFPAGE)
            5 = Seite nach rechts   (WA_RTPAGE)
            6 = Spalte nach links   (WA_LFLINE)
            7 = Spalte nach rechts  (WA_RTLINE)
```

WM_HSLID:

Der Benutzer hat den horizontalen Schieber des Rollbalkens mit der Maus auf eine neue Position geschoben. Die neue Position wird als eine Zahl von 0 bis 1000 angegeben.

```
word 0:     25
word 3:     Die Nummer des Fensters
word 4:     Die verlangte neue Position, wobei ein Wert von 0
            ganz links und ein Wert von 1000 ganz rechts bedeutet
```

WM_VSLID:

Der Schieber des vertikalen Rollbalkens wurde auf eine neue Position gebracht.

```
word 0:     26
word 3:     Die Nummer des Fensters
word 4:     Die neue Position (ebenfalls eine Zahl von 0 bis
            1000), wobei 0 oben und 1000 unten bedeutet.
```

WM_SIZED:

Der Benutzer verlangt eine neue Fenster-Größe (durch Anklicken und verschieben des Feldes rechts unten im Fenster). Die gewünschten neuen Koordinaten des Fensters sind die Äußeren, d. h.

inklusive der Titelzeile, Infozeile und der »Rollbalken (sofern vorhanden). Die neue Arbeitsfläche des Fensters kann mit »*wind_calc()* berechnet werden. Es gibt eine Minimalgröße des Fensters (abhängig vom Vorhandensein von Titelzeile, »Rollbalken und ähnlichem), die der Benutzer nicht unterschreiten kann.

Nun muß das Programm »*wind_set()* aufrufen und die neuen Koordinaten des Fensters übergeben (eventuell nach einer Änderung des Benutzerwunsches). Ebenso muß mit »*wind_set()* die Größe und Position der »Rollbalken (sofern vorhanden) neu gesetzt werden (nach Berechnung derselben).

Wurde das Fenster verkleinert, so braucht es nicht neu gezeichnet werden (eventuell folgen jedoch Nachrichten, die dies für andere Fenster verlangen). Wurde das Fenster vergrößert, so schickt »AES die Aufforderung zum Neuzeichnen.

```
word 0:     27
word 3:     Die Nummer des Fensters
word 4:     Die verlangte X-Koordinate - für gewöhnlich die Alte
word 5:     Die verlangte Y-Koordinate - für gewöhnlich die Alte
word 6:     Die verlangte Breite
word 7:     Die verlangte Höhe
```

WM_MOVED:

Der Benutzer hat das Fenster verschoben. Wenn Sie das Verschieben erlauben wollen, so können Sie diese Anforderung mit »*wind_set()* annehmen. Lag das Fenster teilweise außerhalb des Bildschirms, so folgt dem Aufruf von »*wind_set()* eine Nachricht von »AES, daß das Fenster neu gezeichnet werden muß. Wird das Fenster innerhalb des Bildschirms verschoben, so braucht die Application es nicht neu zu zeichnen; »AES transferiert den Fensterinhalt mittels »Bitblock-Transfer. Um die »Rollbalken brauchen Sie sich nicht zu kümmern das übernimmt »AES.

```
word 0:     28
word 3:     Die Nummer des Fensters
word 4:     Die verlangte neue X-Koordinate
word 5:     Die verlangte neue Y-Koordinate
word 6:     Die alte Breite des Fensters
word 7:     Die alte Höhe des Fensters
```

WM_NEWTOP:

Hiermit teilt »AES mit, daß das Fenster nunmehr aktiviert worden ist - d. h., der Aktivierungsvorgang ist abgeschlossen, das Fenster befindet sich oben (bei WM_TOPPED handelt es sich um die Aufforderung, daß entsprechende Fenster zu aktivieren).

Mit anderen Worten: für dieses Fenster steht die gesamte Arbeitsfläche zum Zeichnen zur Verfügung (abzüglich des Bereichs, der sich nicht mehr auf dem Bildschirm befindet), es wird von keinem anderen Fenster überlappt (höchstens wird es von einem Fenster "unterlappt").

word 0: 29
word 3: Die Nummer des Fensters

Nachrichten für Desk-Accessories

Diese Nachrichten betreffen nur Desk-Accessories, die permanent im Hintergrund laufen, und die daher auch auf eine Nachricht von diesem Typ warten müssen.

AC_OPEN:

Diese Nachricht wird an ein Desk-Accessory geschickt, wenn es aus dem (linken) »Drop-Down-Menü ausgewählt worden ist.

word 0: 30
word 4: Die Identifikationsnummer des Desk-Accessories, die bei der Anmeldung desselben von der Routine »*menu_register()* zurückgegeben worden ist. Hierbei handelt es sich **nicht** um die von »*appl_init()* zurückgegebene Identifikationsnummer (»Task-ID)!
ACHTUNG! Das Wort steht wirklich an fünfter Stelle (Wort Nummer vier) im Message-Puffer!

AC_CLOSE:

Diese Nachricht geht an ein »Desk-Accessory, wenn das gerade ablaufende Hauptprogramm beendet wurde, der Bildschirm gelöscht werden soll und die Strukturen des Window-Managers reinitialisiert werden sollen. Das »Desk-Accessory sollte dann alle von ihm geöffneten Fenster löschen (»*wind_close()* allein reicht nicht, zusätzlich muß auch noch »*wind_delete()* aufgerufen werden). Dies ist z. B. der Fall, wenn ein »TOS-Programm gestartet wird.

word 0: 31
word 4: Die Identifikationsnummer des Desk-Accessories

Timer-Event

Wie die Application auf das Verstreichen eines bestimmten Zeitraums wartet, entnehmen Sie bitte den Beschreibungen von »*evnt_timer()* und »*evnt_multi()*.

MIDI

Abk. für -Musical Instrument Digital Interface-. M. ist zur Kommunikation mit Musikinstrumenten gedacht, die auch eine MIDI-Schnittstelle besitzen. M. arbeit ähnlich wie eine RS 232 Schnittstelle. Die Übertragungsgeschwindigkeit beträgt 31250 Bit pro Sekunde. Sie können die M.-Schnittstelle des Ataris auch für ein Netzwerk mißbrauchen. Bei der M.-Schnittstelle wird grundsätzlich mit einem Startbit, einem Stopbit und 8 Datenbits gearbeitet.

Beim »BIOS hat M. die Gerätenummer drei. Mit der »BIOS-Funktion *Bconstat()* können Sie ermitteln, ob ein Zeichen an der M.-Schnittstelle erhältlich ist, und mit der »BIOS-Funktion *Bcostat()* können Sie ermitteln, ob die M.-Schnittstelle sendebreit ist.

Senden können Sie mit folgenden Funktionen:
»BIOS: *Bconout()*
»XBIOS: *Midiws()*

Empfangen von der M.-Schnittstelle:
»BIOS: *Bconin()*

Mouse-Event

Engl., zu deutsch: Maus-Ereignis. Siehe »Ereignis

Multitasking

M. ist die Fähigkeit eines Rechners, mehrere Programme quasigleichzeitig ablaufen zu lassen, indem zwischen mehreren (multi) Aufgaben (Tasks) umgeschaltet wird.

Auf dem Atari unterstützt das Betriebssystem nur eine eingeschränkte Form des M. Beim »Booten des Betriebssystems können eine Reihe von (bis zu sechs) sog. »Desk-Accessories mitgeladen werden. Dies sind Programme, die resident im Speicher gehalten werden und ständig mitlaufen, d. h. sie warten auf ein »Ereignis. Tritt dieses ein, so gibt das Betriebssystem die Kontrolle an das Accessory ab. Siehe auch »Accessory

Nachricht
siehe »Message

NDC
Abk. für "»Normalized Device Coordinates", zu deutsch normalisierte Geräte-Koordinaten. Siehe »Normalized Device Coordinates

Neuzeichnen von Fenstern
Siehe »Window-Update

Normalisierte Koordinaten
Siehe »Normalized Device Coordinates

Normalized Device Coordinates
Engl., bedeutet soviel wie "Normalisierte Koordinaten". Dabei handelt es sich um ein standardisiertes Koordinatensystem für Grafikdarstellung. Der linke untere Punkt auf dem »Bildschirm wird dabei über das X- Y-Koordinatenpaar 0, 0 angesprochen, der Punkt rechts oben mit 32767, 32767. Dies ist unabhängig von der jeweils verwendeten Auflösung. Die »VDI-Routinen können die N. in die erforderlichen »Raster-Koordinaten umrechnen. Bei jeder Ausgabe wird dies gemacht, gleich, ob sie auf den »Bildschirm, einen Drucker, einen Plotter oder ein beliebiges anderes Grafik-Ausgabegerät geht. Siehe »Raster Koordinaten.

Um NDC-Koordinaten verwenden zu können, **muß** vorher »GDOS installiert worden sein. Dann erst ist es möglich, bei »*v_opnvwk()* eine »virtual Workstation in NDC-Koordinaten zu öffnen.

Objc_add

Routine, die zur »Object-Library gehört.

```
$28 = 40        ob_areturn = objc_add(ob_atree, ob_aparent,
word            ob_areturn,                     ob_aparent;
word *                                ob_atree;
                                      ob_achild)
word                                  ob_achild;
```

Mit dieser Routine können Sie ein »Objekt zu einem »Objektbaum hinzufügen. Präziser gesagt, diese Routine ordnet dem »Elternobjekt ein neues »Kindobjekt (innerhalb desselben Baums!) zu. Das »Resource Construction Set benutzt diese Routine, um Bäume zu konstruieren.

Wenn man nicht genau weiß, wie »Objekte aufgebaut sind, so sollte man besser nicht an »Objektbäumen manipulieren! Die Struktur der »Objektbäume wird durch ein »Resource Construction Set bereits vorher etabliert, so daß es selten nötig ist, dies zur Laufzeit eines Programms zu tun.

Mehr über »Objekte entnehmen Sie bitte dem zugehörigen Stichwort.

Eingabeparameter:

word *ob_atree	Die Adresse des »Objektbaums.
word ob_aparent	Die Indexnummer des »Elternobjekts, dem ein »Kindobjekt zugeordnet werden soll.
word ob_achild	Die Indexnummer des »Kindobjekts.

```
contrl(0)   =  40
contrl(1)   =   2
contrl(3)   =   1

int_in(0)   =  ob_aparent
int_in(1)   =  ob_achild

addr_in(0)  =  ob_atree
```

Ausgabeparameter:

word ob_areturn	0 = Ein Fehler ist geschehen, n = alles in Ordnung.

```
contrl(2)   =   1
contrl(4)   =   0

int_out(0)  =  ob_areturn
```

Objc_change

Routine, die zur »Object-Library gehört.

```
$2F = 47        ob_creturn = objc_change(ob_ctree, ob_cobject,
word            ob_creturn,                        ob_cobject;
word *                                  ob_ctree;
                                        ob_cresvd,
word                                    ob_cresvd;
                                        ob_cxclip, ob_cyclip,
word                                    ob_cxclip, ob_cyclip;
                                        ob_cwclip, ob_chclip,
word                                    ob_cwclip, ob_chclip;
                                        ob_cnewstate, ob_credraw)
word                                    ob_cnewstate, ob_credraw;
```

Dies Routine ändert den Status eines »Objekts und zeichnet es gegebenenfalls neu. Das Ändern des Status heißt, in der Status-Variablen der Objekt-Struktur ein oder mehrere Bits zu setzen oder rückzusetzen und - falls gewünscht - das »Objekt innerhalb des angegebenen Bereichs neu zu zeichnen. Nur falls letzteres gewünscht wird, sollte man diese Routine benutzen. Soll nur die Status-Variable verändert werden, so geht dies auf dem direkten Weg schneller...

Was es mit dem Status eines »Objekts auf sich hat und mit der Objekt-Struktur, können Sie unter "»Objekt" nachlesen!

Eingabeparameter:

word *ob_ctree	Zeiger auf den Objektbaum, in dem sich das zu ändernde »Objekt befindet.
word ob_cobject	Die Indexnummer des »Objekts
word ob_cresvd	Ein vollkommen nutzloser Parameter, der als "reserviert" bezeichnet wird. Er muß aber auf NULL gesetzt werden!
word ob_cxclip	X-Koordinate des Bereichs, in dem das »Objekt neu gezeichnet werden soll - wenn überhaupt...
word ob_cyclip	Y-Koordinate des »Clipping-Bereichs.

word ob_cwclip Breite des »Clipping-Bereichs.

word ob_chclip Das ist die Höhe des »Clipping-Bereichs.

```
contrl(0)   =  47
contrl(1)   =   8
contrl(3)   =   1

int_in(0)   =  ob_cobject
int_in(1)   =  ob_cresvd
int_in(2)   =  ob_cxclip
int_in(3)   =  ob_cyclip
int_in(4)   =  ob_cwclip
int_in(5)   =  ob_chclip

addr_in(0)  =  ob_ctree
```

Ausgabeparameter:

word ob_creturn 0 = Fehler, n = alles Ok.

```
contrl(2)   =  1
contrl(4)   =  0

int_out(0)  =  ob_creturn
```

Objc_delete

Routine, die zur »Object-Library gehört.

```
$29 = 41        ob_dreturn = objc_delete(ob_dltree, ob_dlobject)
word            ob_dreturn,                          ob_dlobject;
word *                                      ob_dltree;
```

Mit dieser Routine können Sie ein »Objekt aus einem »Objektbaum entfernen. Das »Objekt wird nicht physikalisch aus dem Speicher gelöscht, sondern die Zeigerstruktur des »Objektbaums wird umorganisiert!

Eingabeparameter:

word *ob_dltree Die Adresse des »Objektbaums, der das zu löschende »Objekt enthält.

word ob_dlobject Die Indexnummer im Baum des zu löschenden »Objekts.

```
contrl(0)   =  41
contrl(1)   =   1
contrl(3)   =   1

int_in(0)   =  ob_dlobject

addr_in(0)  =  ob_dltree
```

Ausgabeparameter:

```
word ob_dlreturn   0 = Es ist ein Fehler geschehen, n = alles Ok.

contrl(2)   =   1
contrl(4)   =   0

int_out(0)  =  ob_dlreturn
```

Objc_draw

Routine, die zur »Object-Library gehört.

```
$2A = 42        ob_drreturn = objc_draw(ob_drtree, ob_drstartob,
word                ob_drreturn,                        ob_drstartob;
word *                                      ob_drtree;
                                            ob_drdepth, ob_drxclip,
word                                        ob_drdepth, ob_drxclip;
                                            ob_dryclip, ob_drwclip,
word                                        ob_dryclip, ob_drwclip,
                                            ob_drhclip)
word                                        ob_drhclip;
```

Mit dieser Routine können »Objekte auf dem »Bildschirm dargestellt (gezeichnet) werden. Das »Objekt kann z. B. eine »Dialogbox sein, oder ein »Icon - oder ein beliebiges anderes »Objekt. Mit jedem Routinenaufruf wird ein neues Rechteck definiert, innerhalb dessen ausschließlich gezeichnet wird (»Clipping). Ferner kann noch bestimmt werden, ab welcher Ebene der visuellen Hierarchie gezeichnet werden soll, und bei welcher Ebene aufgehört werden soll. Somit lassen sich auch Teile des »Objektbaums darstellen, z. B. die Objekte mit der Indexnummer von drei bis sieben.

Eingabeparameter:

```
word *ob_drtree     Adresse des »Objektbaums, der gezeichnet wer-
                    den soll.

word ob_drstartob   Die Indexnummer des »Objekts, ab dem gezeich-
                    net werden soll.
```

word ob_drdepth — Anzahl der Ebenen des Objektbaums, die gezeichnet werden sollen (ab 'ob_drstartob'). Bei NULL wird beispielsweise nur das Startobjekt dargestellt, bei eins auch noch das nächste usw. - bis zum Ende des Baums, aber auf keinen Fall darüber hinaus! Das letzte »Objekt ist nämlich gekennzeichnet!

word ob_drxclip — Die X-Koordinate des Bereichs, in dem gezeichnet werden soll.

word ob_dryclip — Die Y-Koordinate des Bereichs. Wohlgemerkt, diese Variablen sagen nicht, wohin das »Objekt gesetzt werden soll, sondern in welchem Bereich es dargestellt wird (»Clipping). Die Zeichenkoordinaten sind in der Objektstruktur selbst gespeichert.

word ob_drwclip — Die Breite des »Clipping-Bereichs in Pixeln

word ob_drhclip — Die Höhe des »Clipping-Bereichs in Pixeln

```
contrl(0)   =  42
contrl(1)   =   6
contrl(3)   =   1

int_in(0)   =  ob_drstartob
int_in(1)   =  ob_drdepth
int_in(2)   =  ob_drxclip
int_in(3)   =  ob_dryclip
int_in(4)   =  ob_drwclip
int_in(5)   =  ob_drhclip

addr_in(0)  =  ob_drtree
```

Ausgabeparameter:

word obdrreturn — 0 = Ein Fehler ist vorgekommen, n = alles Ok.

```
contrl(2)   =   1
contrl(4)   =   0

int_out(0)  =  ob_drreturn
```

Objc_edit

Routine, die zur »Object-Library gehört.

```
$2E = 46        ob_edreturn = objc_edit(ob_edtree, ob_edobject,
word            ob_edreturn,                        ob_edobject;
word *                                  ob_edtree;
                                        ob_edchar,
word                                    ob_edchar;
                                        ob_edidx, ob_edkind,
word                                    ob_edidx, ob_edkind;
                                        &ob_ednewidx)
word *                                   ob_ednewidx;
```

Dem Benutzer ermöglicht diese Funktion die Eingabe von Text in einer »Dialogbox. Das »Objekt muß dabei vom Typ G_TEXT oder G_BOXTEXT sein.

Eingabeparameter:

word *ob_edtree	Adresse des »Objektbaums, der das zu editierende Textfeld hat
word ob_edobject	Indexnummer des »Objekts im »Objektbaum
word ob_edchar	Vom Benutzer eingegebenes Zeichen (von der Tastatur gelesen)
word ob_edidx	Die Position des Zeichens bei der Eingabe
word ob_edkind	Art der Aktion, die die Funktion ausführen soll: 0 = reserviert 1 = Eingabemaske aufbauen und Text initialisieren 2 = Text auf dem »Bildschirm anzeigen und im Speicher neu organisieren 3 = Cursor abschalten und das Editieren beenden
word *ob_ednewidx	Zeiger auf Variable für Rückgabewert (s. u.)

```
control(0)  =  46
control(1)  =   4
control(3)  =   1

int_in(0)   =  ob_edobject
int_in(1)   =  ob_edchar
int_in(2)   =  ob_edidx
int_in(3)   =  ob_edkind
```

addr_in(0) = ob_edtree

Ausgabeparameter:

word ob_edreturn 0 = Fehler, n = Alles Ok

word ob_ednewidx Die Position des nächsten Zeichens

contrl(2) = 2
contrl(4) = 0

int_out(0) = ob_edreturn
int_out(1) = ob_ednewidx

Objc_find

Routine, die zur »Object-Library gehört.

```
$2B = 43        ob_fobnum = objc_find(ob_ftree, ob_fstartob,
word            ob_fobnum,                      ob_fstartob;
word *                                  ob_ftree;
                                        ob_fdepth,
word                                    ob_fdepth;
                                        ob_fmx, ob_fmy)
word                                    ob_fmx, ob_fmy;
```

Diese Routine können Sie benutzen, um herauszufinden, welches »Objekt sich gerade unter dem Mauszeiger befindet. Sie geben die X-Y-Koordinaten an, und von wo bis wo im »Objektbaum gesucht werden soll und bekommen die Indexnummer des dort vorgefundenen »Objekts zurück.

Beispiel: Sie haben auf dem Bildschirm mehrere »Icons, und die Funktion »*evnt_multi()* kehrt mit der Meldung zurück, daß ein Mausknopf gedrückt worden ist. Sie geben die Mauskoordinaten (ev. mit »*graf_mkstate()* herausgefunden) an diese Routine sowie die Adresse des »Objektbaums, in dem sich die »Icons befinden, und erhalten die Indexnummer des »Icons zurück, das der Benutzer angeklickt hat.

Eingabeparameter:

word *ob_ftree Die Adresse des »Objektbaums, der abgesucht werden soll. Steht der Mauszeiger über einem »Objekt, das sich aber nicht in diesem Baum befindet, so meldet die Routine "nicht gefunden".

word ob_fstartob Die Indexnummer des »Objekts, ab dem gesucht werden soll. »Objekte mi kleinerer Indexnummer können nicht gefunden werden!

word ob_fdepth Anzahl der Objektebenen, die durchsucht werden sollen. »Objekte auf tieferen Ebenen werden nicht gefunden.

word ob_fmx X-Koordinate, an der nach dort eventuell vorhandenen »Objekten gesucht werden soll. Für gewöhnlich die X-Koordinate des Mauszeigers.

word ob_fmy Y-Koordinate, bei der die Routine suchen soll.

```
contrl(0)   =  43
contrl(1)   =   4
contrl(3)   =   1

int_in(0)   =  ob_fstartob
int_in(1)   =  ob_fdepth
int_in(2)   =  ob_fmx
int_in(3)   =  ob_fmy

addr_in(0)  =  ob_ftree
```

Ausgabeparameter:

word ob_fobnum Die Nummer des »Objekts, das an der angegebenen Stelle gefunden wurde. Bei -1 wurde dort kein »Objekt entdeckt.

```
contrl(2)   =   1
contrl(4)   =   0

int_out(0)  =  ob_fobnum
```

Objc_offset

Routine, die zur »Object-Library gehört.

```
$2C = 44        ob_ofreturn = objc_offset(ob_oftree, ob_ofobject,
word               ob_ofreturn,                      ob_ofobject;
word *                                   ob_oftree;
                                               &ob_ofxoff, &ob_ofyoff)
word *                                          ob_ofxoff,  ob_ofyoff;
```

Mit dieser Routine kann man die Lage eines »Objekts auf dem »Bildschirm relativ zur linken oberen Ecke herausfinden. Die Lage eines »Objekts wird ansonsten immer relativ zum Elternobjekt an-

gegeben. Lediglich die Koordinaten des »Wurzelobjekts beziehen sich auf die linke obere Ecke des »Bildschirms.

Anmerkung: Der Koordinatenursprung ist nur bei »Rasterkoordinaten in der linken oberen Ecke, bei »normalisierten Koordinaten befindet er sich links unten!

Eingabeparameter:

word *ob_oftree	Die Adresse des Objektbaums, in dem sich das »Objekt, dessen Koordinaten berechnet werden sollen, befindet.
word ob_ofobject	Die Indexnummer des »Objekts
word *ob_ofxoff word *ob_ofyoff	Zeiger auf die Rückgabewerte (s. u.).

```
contrl(0)   =  44
contrl(1)   =   1
contrl(3)   =   1

int_in(0)   =  ob_ofobject

addr_in(0)  =  ob_oftree
```

Ausgabeparameter:

word ob_ofreturn	0 = Fehler, n= alles Ok.
word ob_ofxoff	Die X-Koordinate des »Objekts relativ zum Bildschirm (relativ zum Koordinatenursprung).
word ob_ofyoff	Die Y-koordinate des »Objekts

```
contrl(2)   =   3
contrl(4)   =   0

int_out(0)  =  ob_ofreturn
int_out(1)  =  ob_ofxoff
int_out(2)  =  ob_ofyoff
```

Objc_order

Routine, die zur »Object-Library gehört.

```
$2D = 45        ob_orreturn = objc_order(ob_ortree, ob_orobject,
word            ob_orreturn,                         ob_orobject;
word *                                  ob_ortree;
                                        ob_ornewpos)
word                                    ob_ornewpos;
```

Diese Routine bewegt ein »Objekt innerhalb eines »Objektbaums an eine neue Position. Bewegt werden kann ein »Objekt nur innerhalb seines »Elternobjekts. So kann man damit z. B. das zweite »Kind-objekt nehmen und es an die erste Stelle bewegen.

Eingabeparameter:

word *ob_ortree	Adresse des »Objektbaums, in dem sich das zu bewegende »Objekt befindet.
word ob_orobject	Die Indexnummer des »Objekts, welches bewegt werden soll.
word ob_ornewpos	Neue Position für das »Objekt

```
contrl(0)   =  45
contrl(1)   =   2
contrl(3)   =   1

int_in(0)   =  ob_orobject
int_in(1)   =  ob_ornewpos

addr_in(0)  =  ob_ortree
```

Ausgabeparameter:

word ob_orreturn	0 = Fehler, n = alles Ok.

```
contrl(2)   =   1
contrl(4)   =   0

int_out(0)  =  ob_orreturn
```

Object

Siehe »Objekt

Object Library

Bei der O. handelt es sich um eine Sammlung von Routinen zur Manipulation von »Objekten. Sie besteht aus:

- »*objc_add()* Fügt ein »Objekt einem »Objektbaum hinzu
- »*objc_change()* Verändert den Zustand eines »Objekts
- »*objc_delete()* Löscht ein »Objekt aus einem »Objektbaum
- »*objc_draw()* Zeichnet ein »Objekt
- »*objc_edit()* Erlaubt es dem Benutzer, Text in einem »Objekt zu editieren
- »*objc_find()* Findet heraus, welches »Objekt sich an einer bestimmten X-Y-Koordinate befindet
- »*objc_offset()* Berechnet die X-Y-Koordinate eines »Objekts
- »*objc_order()* Ändert die Reihenfolge von Objekten im »Objektbaum

Für weitere Informationen schlage Sie bitte unter dem Stichwort »Objekt und unter den obigen Stichwörtern nach.

Objekt

Ein O. ist ein grafisches Gebilde, welches auf dem »Bildschirm dargestellt werden kann. Ein O. kann also ein Text sein, ein Kasten, eine gefüllte Fläche oder eine Menüleiste. Intern (im Computer) werden diese O. zusammengefaßt in Form eines "Objektbaumes", d. h. einer Struktur, die sich wie ein Baum verzweigt.

Ein Beispiel: Wenn Sie im »Desktop die Menüleiste links oben anwählen und dort den Menüpunkt "Desktop Info", so erscheint in der Mitte des »Bildschirms ein weißer Kasten mit Umrahmung. In diesem Kasten stehen die Copyright-Meldungen von Atari und Digital Research sowie (unten) ein »Button (Knopf).

Der weiße Kasten (die Box) ist das sog. "Wurzelobjekt" - was sich daran erkennen läßt, daß diese Box zuerst gezeichnet wird. In dieser Box befinden sich verschiedene Textzeilen. Diese Textzeilen bezeichnet man als "Abkömmlinge" oder "Kinder". Ferner kann man auch noch einen »Button sehen (unten). Auch dies ist ein Abkömmling des Wurzelobjektes "Box". Jedes Kind kann noch weitere Kinder haben (was in unserem Beispiel nicht der Fall ist). Anders herum gesagt, hat jedes Kind ein Elternteil (nur eines!). Das Elternteil des Wurzelobjekts ist der Bildschirm. Jedes Kind **muß** sich voll und ganz **innerhalb des Elternobjekts** befinden.

Wenn Sie die eben beschriebene Struktur einmal genauer betrachten, so werden Sie feststellen, daß sie einer Baumstruktur ähnlich sieht. Daher also kommt der Ausdruck "Wurzelobjekt"! Und deshalb bezeichnet man das ganze Gebilde auch als "Baum" (»Objektbaum).

Ein weiteres Beispiel für ein Objekt sehen Sie im Anhang A. Was Sie dort sehen, ist die äußere (grafische) Repräsentation des Objekts. Was Sie nicht sehen, ist die interne Repräsentation des Objekts - die Datenstruktur. Diese ist ebenfalls wie ein Baum organisiert dem »Objektbaum. Ein »Objektbaum besteht aus mehreren Objekten. Diese werden in einem Array (Feld) von Objekten zusammengefaßt, in dem jedes Objekt eine Indexnummer (Feldnummer oder Arrayindex) besitzt. Folgende Strukturen können zu einem Objektbaum gehören:
-OBJECT
-TEDINFO
-ICONBLK
-BITBLK
-APPLBLK
-PARMBLK

Im eigentlichen Objektbaum kommen nur Strukturen vom Typ "OBJECT" vor, auf die anderen Objekttypen wird in der OBJECT-Struktur mittels Zeiger verwiesen. Die Benennung leitet sich von den entsprechenden C-Strukuren ab. Die Namen werden auch im folgenden verwendet. Zunächst der Aufbau der Strukturen mit den dazugehörigen Erläuterungen (die "physikalische" Repräsentation, d. h. die Anordnung der Variablen im Speicher):

Offs.	Typ	Variablenname	Erläuterung zur Struktur OBJECT
0	word	ob_next	Indexnummer des nächsten Objekts im Baum
2	word	ob_head	Indexnummer des ersten Kindes dieses O.'s
4	word	ob_tail	Indexnummer des letzten Kindes. Das letzte Objekt zeigt auf das Wurzelobjekt
6	word	ob_type	Typ des Objekts
8	word	ob_flags	Flags für das Objekt
10	word	ob_state	Zustand des Objekts
12	long	ob_spec	Objekt-spezifische Daten (s. u.)
16	word	ob_x	X-Koordinate relativ zum Elternobjekt
18	word	ob_y	Y-Koordinate relativ zum Elternobjekt
20	word	ob_width	Breite des Objekts
22	word	ob_height	Höhe des Objekts

Die OBJECT Struktur beschreibt (durch ihre Indexnummer) die Lage des Objekts in dem Objektbaum, das Verhältnis der Objekte zueinander, den Typ des Objekts (s. u.), den Zustand des Objekts (s. u.), die X- und Y-Koordinaten des Objekts sowie Höhe und Breite. Zunächst die ausführlichen Erläuterungen der Variablen anhand ei-

nes praktischen Beispiels: In der Abbildung im Anhang sehen Sie eine »Dialogbox. Diese besteht aus sechs Objekten, numeriert von Null bis Fünf. Objekt Null ist das Wurzelobjekt, es ist vom Typ "G_BOX" (Grafik-Box). Es hat zwei Abkömmlinge: die Objekte Eins und Zwei. Objekt Eins hat keinen Abkömmling (auch wenn es zunächst anders scheint), Objekt Zwei deren drei: die Objekte Drei, Vier und Fünf. Wenn 'obj_tree' ein Zeiger auf den Objektbaum ist, dann ist 'obj_tree[0]' ein Zeiger auf das Objekt Null; anders gesagt: Null ist die **Indexnummer** des Objekts Null im Objektbaum 'obj_tree'. Mit dieser Indexnummer kann man also auf einzelne Objekte im Baum zugreifen. Die "Nachfolgeregelung" (welches Kind gehört zu welchem Elternteil?) geschieht nach der Art der verketteten Liste. Die Abbildung im Anhang A zeigt für die Beispiel-Dialogbox den Zusammenhang zwischen Variablenwerten und Baumstruktur. Die Verkettung erfolgt über die Indexnummern der Objekte im Objektbaum (die Punkte deuten an, daß in der Objektstruktur noch ein paar Werte fehlen).

Beispiel: 'ob_next' ist die Indexnummer des nächsten Objekts **das sich auf der gleichen oder einer tieferen Stufe** befindet. Zwei Objekte befinden sich auf der gleichen Stufe, wenn sie ein gemeinsames Elternobjekt haben. Ein Wert von -1 besagt, daß kein solches Objekt vorhanden ist. In unserem Beispiel hat der äußere Kasten (Objekt Null), da es das Wurzelobjekt ist, nur den Bildschirm als Elternobjekt. Bei Objekt Eins hat 'ob_next' den Wert zwei - die Indexnummer des nächsten Objekts. Das nächste Objekt für Objekt Drei hat die Nummer Vier usw. Das letzte Objekt des Baums weist immer zum Anfang zurück: Objekt Fünf weist mit 'ob_next' auf Objekt Zwei und dies wiederum auf das Wurzelobjekt.

'Ob_head' zeigt immer auf den nächsten Abkömmling. Die Objekte Drei, Vier und Fünf markieren das Ende der Verzweigung: diese Objekte haben keine Kinder. 'Ob_tail' enthält die Indexnummer des **letzten** Kindes, das zum Objekt gehört. Sehen Sie sich dazu die entsprechende Abbildung im Anhang A an!

Zugegeben: einfach ist das nicht, aber... effektiv. Es erscheint zunächst umständlich, offenbart aber mit der Zeit eine gewisse Raffinesse. Viele der Feinheiten muß man nicht wissen, um sie verwenden zu können - aber man sollte wissen, wo sie stehen, falls man sie doch mal braucht!

"Moment! Wieso ist die Zeichenkette in Objekt Zwei ein eigenständiges Objekt, der Text in Objekt Eins aber nicht?" Nun, das hängt damit zusammen, daß Objekt Eins vom Typ "G_BOXTEXT" ist, der Text zum Objekt also dazugehört. Womit wir beim Thema Objekttypen wären und damit bei der Variablen 'ob_type'. Hier eine Tabelle der existierenden Typen:

Objekttyp	Nr.	Erläuterung
G_BOX	20	Ein mit Muster gefülltes Rechteck
G_TEXT	21	Grafiktext (kann zwei verschiedene Größen haben)
G_BOXTEXT	22	Wie G_TEXT, aber von einem Kasten umrahmt
G_IMAGE	23	Ein »Bit-Image, also ein Grafik-Bild
G_PROGDEF	24	Objekt enthält die Adresse einer Zeichenroutine
G_IBOX	25	Durchsichtiges (nur aus Rahmen bestehendes) Rechteck
G_BUTTON	26	Rechteck mit Text als wählbarer Knopf
G_BOXCHAR	27	Wie G_BOXTEXT, enthält aber nur ein Zeichen
G_STRING	28	Grafiktext (kann nur eine Größe haben)
G_FTEXT	29	Formatierter Grafiktext (z. B. zentriert)
G_FBOXTEXT	30	Kasten mit formatiertem Grafiktext
G_ICON	31	Ein »Icon, ein »Bit-Image mit Maske
G_TITLE	32	Grafiktext für »Drop-Down-Menüs

Wie man sieht: reiche Auswahl, nicht einmal das vom Benutzer selbst definierbare Objekt (G_PROGDEF) fehlt! Die Namen und die Nummern der Objekttypen stammen aus OBDEFS.H und sind natürlich besonders für die Programmierer des hohen C interessant.

So ein Objekt kann natürlich noch verschiedene Eigenschaften haben. Diese werden in 'ob_flags' festgelegt (und zwar bitweise):

Bit	symb. Name	Erläuterung
0	SELECTABLE	Das Objekt kann vom Benutzer (durch Anklicken) ausgewählt werden
1	DEFAULT	Dieses Objekt kann auch durch <Return> gewählt werden (gewöhnlich ein EXIT-Button); **nur ein Objekt** darf dieses Flag haben (in einem Objektbaum)!
2	EXIT	Bei »Dialogboxen führt das Anklicken dieses Objekts zum Beenden des Dialogs
3	EDITABLE	Objekt besteht aus einem editierbaren Textfeld
4	RBUTTON	Ein Objekt, das nur rudelweise auftritt: wird eines davon selektiert, werden alle anderen der Gruppe automatisch deselektiert (RadioBUTTON, wie die Senderknöpfe eines Radios) die verschiedenen RBUTTON's haben dasselbe Elternobjekt
5	LASTOB	Kennzeichnet das letzte Objekt in der Baumstruktur
6	TOUCHEXIT	Dieses Objekt gilt als ausgewählt, wenn der Benutzer es anklickt - **sonst** wird die Wahl erst vollzogen, wenn der Mausknopf **losgelassen** wird **und der Mauszeiger sich innerhalb des Objekts befindet**

Bit	symb. Name	Erläuterung
7	HIDETREE	Objekt wird nicht mitgezeichnet
8	INDIRECT	'ob_spec' ist ein Zeiger auf den tatsächlichen Wert von 'ob_spec'

Die Namen entstammen wieder "OBDEFS.H". Sie werden sie auch in den Bezeichnungen der Abbildung im Anhang A wiederfinden.

Im folgenden finden Sie die verschiedenen Zustände, die ein Objekt annehmen kann. Der Zustand wird in der Variablen 'ob_state' angegeben. Wenn Sie z. B. den linken Knopf im Beispiel-Dialog anklicken, so wird in der Variablen 'ob_state' das Bit Null gesetzt (Zustand = SELECTED). Bit fünf ist übrigens auch (von vornherein) gesetzt. Nach allgemeiner Konvention bedeutet dies, daß bei 'ob_flags' Bit eins gesetzt ist ("DEFAULT"). Letzteres wiederum bedeutet, daß Sie dieses Objekt durch Drücken der <Return>-Taste auswählen können. Welche der »Buttons der Benutzer ausgewählt hat, können Sie herausfinden, wenn Sie Bit Null von 'ob_state' testen. **Wenn** also ein Objekt wählbar ist (Bit Null -SELECTABLE - bei 'ob_flags' gesetzt) **und** Bit Null bei 'ob_state' (=SELECTED) gesetzt, **dann** wurde dieses Objekt vom Benutzer gewählt.

Was hat es mit 'ob_spec' auf sich? Nun, diese Variable hat je nach Typ des Objekts (s. o.) eine andere Bedeutung. Entweder handelt es sich um einen Zeiger auf etwas, oder die vier Bytes enthalten irgendwelche Werte. Ist letzteres der Fall, so bezeichne ich mit Byte Null die unteren acht Bit (Bit Null bis sieben), mit Byte Eins die Bits acht bis 15, mit Byte Zwei die Bits 16 bis 23 und mit Byte Drei vom oberen Wort die oberen acht Bits (Bit 24 bis 31). Im folgenden sehen Sie eine tabellarische Aufstellung der Bedeutung:

Objekttyp	Bedeutung von 'ob_spec'
G_BOX	Byte Null und Byte Eins: Farbe des Objekts Byte Zwei: enthält die Dicke des Objektrandes: 0 = kein Rand 1 - 128 = (positiver Wert) Dicke des Randes in »Pixeln, nach innen zählend von den Ecken des Objekts -1 - (-127) = (negativer Wert) Dicke des Randes in »Pixeln, nach außen zählend von den Ecken des Objekts Byte drei = Null
G_TEXT	Zeiger auf die zugehörige TEDINFO-Struktur, in der die Merkmale des Textes festgelegt werden (s. u.)
G_BOXTEXT	wie bei G_TEXT
G_IMAGE	Zeiger auf Struktur vom Typ ICONBLK, in der das »Bit-Image beschrieben wird

Objekttyp	Bedeutung von 'ob_spec'
G_PROGDEF	Zeiger auf eine Struktur vom Typ APPLBLK, die das Benutzerdefinierte Objekt beschreibt (APPLBLK enthält die Adresse der Routine, die das Objekt zeichnet)
G_IBOX	wie bei G_BOX
G_BUTTON	wie bei G_TEXT
G_BOXCHAR	wie bei G_BOX, aber Byte Drei enthält das darzustellende Zeichen
G_STRING	Zeiger auf den String selbst
G_FTEXT	wie bei G_TEXT
G_FBOXTEXT	wie bei G_TEXT
G_ICON	Zeiger auf eine Struktur vom Typ ICONBLK, in der das »Icon genauer beschrieben ist
G_TITLE	Zeiger auf den Text für den Menüeintrag

Wie man sieht: der Objektbaum verzweigt sich noch weiter! In unserem Beispiel 'ob_spec' von Objekt Eins auf eine TEDINFO-Struktur zeigt, und 'ob_spec' für Objekt Drei auf einen String. Wobei wir noch hinzufügen möchten, daß diese Beschreibung nur für Leute interessant ist, die es ganz genau wissen möchten. Die Arbeit, einen solchen Objektbaum aufzubauen, wird Ihnen nämlich von einem »Resource Construction Set abgenommen! Wie man diese Erkenntnisse verwendet, um z. B. »Dialogboxen zu erzeugen, ist unter dem entsprechenden Stichwort nochmals genau beschrieben.

Die TEDINFO-Struktur: sie wird benötigt, um Text in einem Objekt genauer zu beschreiben. Dies wird notwendig, wenn es sich um zentrierten und/oder kleingeschriebenen Text handelt oder um editierbare Textfelder. Dann reicht es nicht wie bei G_STRING aus, in der OBJECT-Struktur nur einen Zeiger auf den String selbst zu liefern.

Offs.	Typ	Variablenname	Erläuterung z. Struktur TEDINFO
0	long	te_ptext	Zeiger auf Text
4	long	te_ptmplt	Zeiger auf Eingabe-Maske
8	long	te_pvalid	Zeiger auf erlaubte Zeichen für Eingabe
12	word	te_font	Benutzter Zeichensatz
14	word	te_junk1	Ohne Bedeutung
16	word	te_color	Enthält die Zeichenfarbe
18	word	te_junk2	Ohne Bedeutung
20	word	te_thickness	Dicke der Randlinie
22	word	te_txtlen	Länge des Textstrings te_ptext
24	word	te_tmplen	Länge der Eingabemaske

Die Variable 'te_ptext' ist ein Zeiger auf den tatsächlichen »ASCII-Text, der mit einem Null-Byte abgeschlossen wird (wie in C üblich). Wenn das erste Zeichen ein '@' ist, so ist der gesamte String als leer anzusehen; "@xxxx" wäre ein String mit fünf Leer-

zeichen. Die Gesamtlänge der Zeichenkette ist aber **sechs**, denn das Null-Byte **muß** mitgerechnet werden.

Bei 'te_ptmplt' handelt es sich um eine Eingabemaske (genauer beschrieben unter dem Stichwort »Dialogbox).

Offs.	Typ	Variablenname	Erläuterung zur Struktur ICONBLK
0	long	ib_pmask	Zeiger auf das Array mit Maskendaten
4	long	ib_pdata	Zeiger auf das Array mit den Daten
8	long	ib_ptext	Zeiger auf den »Icon-Text
12	word	ib_char	Zeichen für »Icon
14	word	ib_xchar	X-Koordinate für das Zeichen 'ib_char'
16	word	ib_ychar	Y-Koordinate für das Zeichen 'ib_char'
18	word	ib_xicon	X-Koordinate des »Icon
20	word	ib_yicon	Y-Koordinate des »Icon
22	word	ib_wicon	Breite des »Icon
24	word	ib_hicon	Höhe des »Icon
26	word	ib_xtext	X-Koordinate des Icon-Textes
28	word	ib_ytext	Y-Koordinate des Icon-Textes
30	word	ib_wtext	Breite des »Icon-Textes
32	word	ib_htext	Höhe des »Icon-Textes

Ein »Icon besteht aus einem (meistens) 16 * 16 »Pixeln großen »Bit-Image (größere oder kleinere sind möglich) und einer Maske. Die Maskendaten werden verwendet, um eine invertierte Darstellung des »Icons erzielen zu können (dies geschieht meistens, wenn ein »Icon angeklickt wird). Die Maskendaten werden mit den Icondaten durch eine bitweise exklusiv-Oder-Operation verknüpft (XOR). Ferner besteht das »Icon noch aus einem Buchstaben (bei den Laufwerk-Icons auf dem »Desktop der Buchstabe des Laufwerks) und einem Textstring (in Kleinschrift, Größe eines Buchstabens: 6 * 6 »Pixel). Die Parameter für die Darstellung finden Sie in der Beschreibung der Struktur oben wieder.

Wie »Icons in Ihrem Programm verwendet werden können, lesen Sie unter dem entsprechenden Stichwort nach.

Falls Sie ein kleines Bild darstellen wollen, ohne daß dieses wählbar sein soll, so wäre die Verwendung eines »Icons Platzverschwendung (Maskendaten und Text werden ja nicht benötigt). Für diesen Fall nimmt man ein ein Objekt vom Typ "G_IMAGE". Dessen 'ob_spec'-Variable zeigt auf eine Struktur vom Typ "BITBLK", die hier erklärt wird:

Offs.	Typ	Variablenname	Erläuterung zur Struktur BITBLK
0	long	bi_pdata	Zeiger auf das Array mit den Daten
4	word	bi_wb	Breite des Bitblocks in Wörtern
6	word	bi_hl	Höhe in Rasterzeilen (Scanlines)
8	word	bi_x	X-Koordinate des Bitblocks
10	word	bi_y	Y-Koordinate des Bitblocks
12	word	bi_color	Vordergrund-Farbe des Bitblocks

Hierzu muß nur noch gesagt werden, daß "Höhe in Rasterzeilen" dasselbe bedeutet wie "Höhe in »Pixeln".

Die nächste Struktur, die uns interessiert, heißt "APPLBLK". Sie wird verwendet, um vom Programmierer definierte Objekte darstellen zu können. Nehmen wir an, Sie wollen zur Abwechslung mal eine runde »Dialogbox darstellen. Sie werden kein vordefiniertes Objekt finden, das dieses Form hat (außer, Sie verwenden ein Objekt vom Typ "G_IMAGE" und definieren es selbst). Dann können Sie eine Routine schreiben (in C, Assembler, Pascal oder einer anderen Sprache, die Maschinencode erzeugt), die einen Kreis zeichnet (z. B. mit der »VDI-Routine »*v_circle()*). Sie ändern dann im Programm den Typ des äußeren Objekts auf "G_PROGDEF" um und lassen 'ob_spec' auf eine Struktur vom Typ "APPLBLK" zeigen (s. u.). In diese Struktur tragen Sie die Adresse Ihrer Zeichenroutine ein, sowie eventuell einen Parameter (falls gewünscht). Zum Zeichnen des Objekts "G_PROGDEF" wird diese Routine aufgerufen.

"G_PROGDEF" kann auch dazu verwendet werden, den Zustand eines Objekts zu ändern (auf die gleiche Art wie oben beschrieben).

Offs.	Typ	Variablenname	Erläuterung zur Struktur APPLBLK
0	long	ub_code	Zeiger auf eine Funktion (Adresse der Routine)
4	long	ub_parm	Optionaler Parameter für die Routine

Wenn nun die Routine aufgerufen wird, auf die 'ub_code' zeigt, so wird ein Zeiger auf eine Struktur namens "PARMBLK" mitgeliefert. Damit erhält die Routine Informationen über das Objekt, das gezeichnet oder geändert werden soll. Die Struktur sieht so aus:

Offs.	Typ	Variablenname	Erläuterung zur Struktur PARMBLK
0	long	pb_tree	Zeiger auf die OBJECT-Struktur
4	word	pb_obj	Index des anwendungsspezifischen Objekts in dem »Objektbaum, auf den 'pb_tree' zeigt
6	word	pb_prevstate	Alter Zustand des zu ändernden Objekts

Offs.	Typ	Variablenname	Erläuterung zur Struktur PARMBLK
8	word	pb_currstate	Veränderter Zustand des Objekts (wenn 'pb_prevstate' = 'pb_currstate' dann wird das Objekt gezeichnet und nicht geändert)
10	word	pb_x	X-Koordinate des Objekts (relativ zum Elternobjekt!)
12	word	pb_y	Y-Koordinate des Objekts
14	word	pb_w	Breite des Objekts
16	word	pb_h	Höhe des Objekts
18	word	pb_xc	X-Koordinate des Rechtecks für »Clipping
20	word	pb_yc	Y-Koordinate des Rechtecks für »Clipping
22	word	pb_wc	Breite des Rechtecks für »Clipping
24	word	pb_hc	Höhe des Rechtecks für »Clipping
26	long	pb_parm	Identisch mit 'ub_parm' von APPLBLK

Die Daten für das »Clipping werden beim Aufruf der Routine »*objc_draw()* mit angegeben.

Unter dem Stichwort »Dialogbox finden Sie ein praktisches Beispiel für die Verwendung von Objekten. Weiterhin gibt es zur Manipulation von Objekten eine Reihe von Routinen. Diese sind unter dem Eintrag »Object Library erklärt.

Objektbaum

Ein O. ist die Zusammenfassung von mehreren Objektstrukturen in einem Array. Auf die einzelnen »Objekte kann über eine Indexnummer (Feld- oder Arrayindex) zugegriffen werden. Die (logische) Anordnung der Strukturen entspricht einem Baum. Mehr darüber bei der Erläuterung der »Objekte.

Offset

Allgemein wird mit O. ein Abstand von einer bestimmten Marke bezeichnet. Für die CPU »MC 68000 gibt es sowohl 8, als auch 16 Bit große Offsets. Bei den Branchbefehlen ist der 8 Bit Offset direkt mit im »Opcode codiert. Hat er den Wert Null, dann wird das nächste Wort im Speicher als 16 Bit Offset benutzt. Bei Adressierungsarten, die mit Offsets (Distanzen) arbeiten, wird immer das nächste Wort im Speicher als Offset benutzt. Geht aus dem »Opcode hervor, daß es sich um einen 8 Bit Offset handelt, dann werden nur die Bits 0 bis 7 des nächsten Wortes benutzt. Alle Offsets werden im »Zweierkomplement angegeben. Der Offset als Adreßdi-

stanz bei der Adressierung wird direkt zum Inhalt des Adreßregisters addiert. Bei Branchbefehlen wird er zum aktuellen Inhalt des Programmzählers addiert, d.h. zur Adresse, die direkt hinter dem Branchbefehl steht.

Vorwärts

	$.0	$.1	$.2	$.3	$.4	$.5	$.6	$.7	$.8	$.9	$.A	$.B	$.C	$.D	$.E	$.F
$0.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$1.	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$2.	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$3.	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$4.	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
$5.	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
$6.	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
$7.	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127

Rückwärts

	$.0	$.1	$.2	$.3	$.4	$.5	$.6	$.7	$.8	$.9	$.A	$.B	$.C	$.D	$.E	$.F
$8.	128	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113
$9.	112	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97
$A.	96	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81
$B.	80	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65
$C.	64	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49
$D.	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33
$E.	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17
$F.	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

Opcodes

Die Opcodes sind die Worte, die die CPU als Befehl einliest und ausführen kann. Die CPU »MC68000 erkennt fast alle Kombinationen der 16 Bit Worte als Befehle an. Die illegalen Opcodes sind Kombinationen, die das Mikroprogramm der CPU nicht ausführen kann. Stattdessen wird dann die »Exception *Illegal Instruction* über den Vektor 4 in Adresse $000010 ausgeführt. Die von der CPU bearbeitbaren Opcodes finden Sie in der folgenden Tabelle. Links steht das »Mnemonic des Befehls, dann folgt dessen Größe. Dabei bedeutet: **B**=Byte (8 Bit), **W**=Word (16 Bit) und **L**=Long (32 Bit). Die **Adressierung** gibt die Quelle (s=source) und das Ziel (d=destination) des Befehls an.

Allgemein gilt:

A_n = Eines der Adreßregister von 0 bis 7.
D_n = Eines der Datenregister von 0 bis 7.
PC = Programmzähler.
X_n = Daten- oder Adreßregister.

Die Ziffern haben folgende Bedeutung:

1 = D_n	2 = A_n	3 = (A_n)	4 = $(A_n)+$
5 = $-(A_n)$	6 = $d_{16}(A_n)$	7 = $d_8(A_n,X_n)$	8 = Abs.W
9 = Abs.L	A = $d_{16}(PC)$	B = $d_8(PC,X_n)$	C = Imm/SR/CC

Die Zahlen in den darunterliegenden Spalten geben die Anzahl an Bytes an, die dieser Befehl lang ist. Steht bei einer bestimmten Adressierungsart ein Strich '-', dann gibt es diesen Befehl in dieser Art nicht.

Daran schließen sich die Flaggen an:

X = Extend N = Negativ Z = Zero V = Overflow
C = Carry

Ein Stern '*' in der Spalte bedeutet, daß diese Flagge entsprechend dem Resultat dieses Befehls gesetzt wird. Ein Fragezeichen '?' zeigt an, daß der Zustand dieser Flagge nach dem Befehl unbestimmt ist! Bei einem Tildezeichen '~' wird die Flagge vom Befehl nicht verändert. Eine Eins '1' zeigt eine durch den Befehl gesetzte Flagge an und eine Null '0' eine gelöschte.

In der ganz rechten Spalte folgen die Opcodes in bitweiser Codierung. In diesem Feld bedeuten Buchstaben:

AAA = Nummer eines Adreßregisters im Bereich [0;7].

CCCC = Verzweigungsbedingung. Es gilt folgende Codierung:

Code	Mnemo	Bedingung	Test auf:
%0000	T	true	1
%0001	F	false	0
%0010	HI	higher than	$\bar{C}\wedge\bar{Z} = 1$
%0011	LS	less than	$C\vee Z = 1$
%0100	CC/HS	higher or same	$C = 0$
%0101	CS/LO	lower than	$C = 1$
%0110	NE	not equal	$Z = 0$
%0111	EQ	equal	$Z = 1$
%1000	VC	overflow clear	$V = 0$
%1001	VS	overflow set	$V = 1$
%1010	PL	plus	$N = 0$
%1011	MI	minus	$N = 1$
%1100	GE	greater or equal	$(N\wedge V)\vee(\bar{N}\wedge\bar{V}) = 1$
%1101	LT	less than	$(N\wedge\bar{V})\vee(\bar{N}\wedge V) = 1$
%1110	GT	greater than	$(N\wedge V\wedge\bar{Z})\vee(\bar{N}\wedge\bar{V}\wedge\bar{Z}) = 1$
%1111	LE	less or equal	$(N\wedge V)\vee(\bar{N}\wedge\bar{V})\vee Z = 1$

DDD = Nummer eines Datenregisters im Bereich [0;7].

EEEEEE = Effektive Adresse des Ziels. Eine effektive Adresse hat eine Länge von 6 Bits. Die unteren drei geben die Nummer des benutzten Registers an und die oberen drei:

Code	Adressierungsart
%000.rrr	D_n = Datenregister direkt
%001.rrr	A_n = Adreßregister direkt
%010.rrr	(A_n) = Adreßregister indirekt (ARI)
%011.rrr	$(A_n)+$ = ARI mit Postinkrement
%100.rrr	$-(A_n)$ = ARI mit Predekrement
%101.rrr	$d_{16}(A_n)$ = ARI mit Adreßdistanz
%110.rrr	$d_8(A_n,X_n)$ = ARI mit Distanz und Index
%111.000	$xxxx = Absolut kurz (Word)
%111.001	$xxxxxxxx = Absolut lang (Long)
%111.010	$d_{16}(PC)$ = PC relativ mit Distanz
%111.011	$d_8(PC,X_n)$ = PC relativ mit Distanz und Index
%111.100	#/SR/CCR = Immediate/Statusreg./Conditionreg.
%111.101	nicht benutzt
%111.110	nicht benutzt
%111.111	nicht benutzt

MMM = Modus der effektiven Adressierung des Zieles.
PPPPPPPP = »Sprungweite bei Branchbefehlen. Ist dieser Wert gleich Null, dann handelt es sich um einen 16 Bit Offset, der im folgenden Wort steht.
QQQ = 3 Bit Wert für Immediate Daten.
RRR = Nummer des Zielregisters.
SS = Größe der Operanden: %00=Byte, %01=Word, %10=Long.
VVVV = Nummer des Vektors von 0 bis 15.
XX = Operandengröße beim **MOVE**-Befehl: %01=Byte, %11=Word.
eeeeee = Effektive Adresse der Quelle.
f = Richtung: 0=rechts, 1=links.
rrr = Nummer des Quellregisters.

Mnemo	Size	Adressierung	1234567890AB	XNZVC	FEDC	BA98	7654	3210
ABCD	B	s=D_n	d=2----------	*?*?*	1100	RRR1	0000	0rrr
		s=-(A_n)	d=----2-------		1100	RRR1	0000	1rrr
ADD	B/W	s=D_n	d=2-2224446---	*****	1101	DDD1	SSEE	EEEE
		d=D_n	s=222224446444		1101	DDD0	SSee	eeee
	L	s=D_n	d=2-2224446---		1101	DDD1	10EE	EEEE
		d=D_n	s=222224446446		1101	DDD0	10ee	eeee
ADDA	W	d=A_n	s=222224446444	~~~~~	1101	AAA0	11ee	eeee
	L	d=A_n	s=222224446446		1101	AAA1	11ee	eeee
ADDI	B/W	s=Imm	d=4-4446668---	*****	0000	0110	SSEE	EEEE
	L	s=Imm	d=6-666888A---					
ADDQ	B/W	s=Imm3	d=222224446---	*****	0101	QQQ0	SSEE	EEEE
	L	s=Imm3	d=222224446---					
ADDX	B/W	s=D_n	d=2----------	*****	1101	RRR1	SS00	0rrr
		s=-(A_n)	d=----2-------		1101	RRR1	SS00	1rrr
	L	s=D_n	d=2----------		1101	RRR1	1000	0rrr
		s=-(A_n)	d=----2-------		1101	RRR1	1000	1rrr

AND	B/W	s=D_n	d=2-2224446---	~**00	1100	DDD1	SSEE	EEEE
		d=D_n	s=2-2224446444		1100	DDD0	SSee	eeee
	L	s=D_n	d=2-2224446---		1100	DDD1	10EE	EEEE
		d=D_n	s=2-2224446446		1100	DDD0	10ee	eeee
ANDI	B/W	s=Imm	d=4-4446668---	~**00	0000	0010	SSEE	EEEE
	L	s=Imm	d=6-666888A---					
ANDI CCR	B	s=Imm	d=----------4	*****	0000	0010	0011	1100
ANDI SR	W	s=Imm	d=----------4	*****	0000	0010	0111	1100
ASL	B/W	count=D_n	d=2----------	*****	1110	rrrf	SS10	0DDD
ASR		count=#1-8	d=2----------		1110	QQQf	SS00	0DDD
	L	count=D_n	d=2----------		1110	rrrf	1010	0DDD
		count=#1-8	d=2----------		1110	QQQf	1000	0DDD
Memory	W	count=1	d=--2224446---		1110	000f	11EE	EEEE
Bcc	B		d_8=----------2	~~~~~	0110	CCCC	PPPP	PPPP
	W		d_{16}=----------4					
BCHG	B	bit#=D_n	d=--2224446---	~~*~~	0000	rrr1	01EE	EEEE
		bit#=Imm	d=--4446668---		0000	1000	01EE	EEEE
	L	bit#=D_n	d=2----------		0000	rrr1	01EE	EEEE
		bit#=Imm	d=4----------		0000	1000	01EE	EEEE
BCLR	B	bit#=D_n	d=--2224446---	~~*~~	0000	rrr1	10EE	EEEE
		bit#=Imm	d=--4446668---		0000	1000	10EE	EEEE
	L	bit#=D_n	d=2----------		0000	rrr1	10EE	EEEE
		bit#=Imm	d=4----------		0000	1000	10EE	EEEE
BRA	B		d_8=----------2	~~~~~	0110	0000	PPPP	PPPP
	W		d_{16}=----------4					
BSET	B	bit#=D_n	d=--2224446---	~~*~~	0000	rrr1	11EE	EEEE
		bit#=Imm	d=--4446668---		0000	1000	11EE	EEEE
	L	bit#=D_n	d=2----------		0000	rrr1	11EE	EEEE
		bit#=Imm	d=4----------		0000	1000	11EE	EEEE
BSR	B		d_8=----------2	~~~~~	0110	0001	PPPP	PPPP
	W		d_{16}=----------4					
BTST	B	bit#=D_n	d=--2224446---	~~*~~	0000	rrr1	00EE	EEEE
		bit#=Imm	d=--4446668---		0000	1000	00EE	EEEE
	L	bit#=D_n	d=2----------		0000	rrr1	00EE	EEEE
		bit#=Imm	d=4----------		0000	1000	00EE	EEEE
CHK	W	d=D_n	s(lim)=2-2224446444	~*???	0100	DDD1	10ee	eeee
CLR	B/W		d=2-2224446---	~0100	0100	0010	SSEE	EEEE
	L		d=2-2224446---					
CMP	B/W	d=D_n	s=222224446444	~****	1011	DDD0	SSee	eeee
	L	d=D_n	s=222224446446					
CMPA	W	d=A_n	s=222224446444	~****	1011	AAA0	11ee	eeee
	L	d=A_n	s=222224446446		1011	AAA1	11ee	eeee
CMPI	B/W	s=Imm	d=4-4446668---	~****	0000	1100	SSEE	EEEE
	L	s=Imm	d=6-666888A---					
CMPM	B/W	s=(A_n)+	d=---2--------	~****	1011	RRR1	SS00	1rrr
	L	s=(A_n)+	d=---2--------					
DBcc	W	d_{16}=Imm	d=4----------	~~~~~	0101	CCCC	1100	1DDD
DIVS	W	d=D_n	s=2-2224446444	~***0	1000	DDD1	11ee	eeee
DIVU	W	d=D_n	s=2-2224446444	~***0	1000	DDD0	11ee	eeee

EOR	B/W	s=D_n	d=2-2224446---	~**00	1011 rrr1 SSEE EEEE
	L	s=D_n	d=2-2224446---		
EORI	B	s=Imm	d=4-4446668---	~**00	0000 1010 SSEE EEEE
	L	s=Imm	d=6-666888A---		
EORI CCR	B	s=Imm	d=----------4	*****	0000 1010 0011 1100
EORI SR	W	s=Imm	d=----------4	*****	0000 1010 0111 1100
EXG	L	s=D_n	d=22----------	~~~~~	1100 DDD1 0100 0DDD
		s=a_n	d=22----------		1100 AAA1 0100 1AAA
					1100 DDD1 1000 1AAA
EXT	W		d=2----------	~**00	0100 1000 1000 0DDD
	L		d=2----------		0100 1000 1100 0DDD
ILLEGAL			2----------	~~~~~	0100 1010 1111 1100
JMP			d=--2--444644-	~~~~~	0100 1110 11EE EEEE
JSR			d=--2--444644-	~~~~~	0100 1110 10EE EEEE
LEA	L	d=A_n	s=--2--444644-	~~~~~	0100 AAA1 11ee eeee
LINK		d_{16}=Imm	s=-4----------	~~~~~	0100 1110 0101 0AAA
LSR	B/W	count=D_n	d=2----------	***0*	1110 rrrf SS10 1DDD
LSL		count=#1-8	d=2----------		1110 QQQf SS00 1DDD
	L	count=D_n	d=2----------		1110 rrrf 1010 1DDD
		count=#1-8	d=2----------		1110 QQQf 1000 1DDD
Memory	W	count=1	d=--2224446---		1110 001f 11EE EEEE
MOVE	B/W	d=D_n	s=222224446444	~**00	00XX RRRM MMee eeee
		d=A_n	››››siehe MOVEA‹‹‹		
		d=(A_n)	s=222224446444		
		d=(A_n)+	s=222224444444		
		d=-(A_n)	s=222224446444		
		d=d_{16}(A_n)	s=444446668666		
		d=d_8(A_n,X_n)	s=444446668666		
		d=ABS.W	s=444446668666		
		d=ABS.L	s=66666888A888		
MOVE	L	d=D_n	s=222224446446	~**00	0100 RRRM MMee eeee
		d=A_n	››››siehe MOVEA‹‹‹		
		d=(A_n)	s=222224446446		
MOVE CCR	W	d=CCR	s=2-2224446444	*****	0100 0100 11ee eeee
MOVE SR	W	d=SR	s=2-2224446444	*****	0100 0110 11ee eeee
		s=SR	d=2-2224446---	~~~~~	0100 0000 11EE EEEE
MOVE USP	L	s=USP	d=-2----------	~~~~~	0100 1110 0110 1AAA
		d=USP	s=-2----------		0100 1110 0110 0AAA
MOVEA	W	d=A_n	s=222224446444	~~~~~	0011 AAA0 01ee eeee
	L	d=A_n	s=222224446446		0010 AAA0 01ee eeee
MOVEM	W	s=X_n	d=--4-46668---	~~~~~	0100 1000 10EE EEEE
		d=X_n	s=--44-666866-		0100 1100 10ee eeee
	L	s=X_n	d=--4-46668---		0100 1000 11EE EEEE
		d=X_n	s=--44-666866-		0100 1100 11ee eeee
MOVEP	W	s=D_n	d=-----4------	~~~~~	0000 DDD1 1000 1AAA
		s=d_{16}(A_n)	d=4----------		0000 DDD1 0000 1AAA
	L	s=D_n	d=-----4------		0000 DDD1 1100 1AAA
		s=d_{16}(A_n)	d=4----------		0000 DDD1 0100 1AAA
MOVEQ	L	s=Imm_8	d=2----------	~**00	0111 DDD0 QQQQ QQQQ

MULS	W	d=Dn	s=2-2224446444	~**00	1100	DDD1	11ee	eeee
MULU	W	d=Dn	s=2-2224446444	~**00	1100	DDD0	11ee	eeee
NBCD	B		d=2-2224446---	*?*?*	0100	1000	00EE	EEEE
NEG	B/W		d=2-2224446---	*****	0100	0100	SSEE	EEEE
	L		d=2-2224446---					
NEGX	B/W		d=2-2224446---	*****	0100	0000	SSEE	EEEE
	L		d=2-2224446---					
NOP			2----------	~~~~~	0100	1110	0111	0001
NOT	B/W		d=2-2224446---	~**00	0100	0110	SSEE	EEEE
	L		d=2-2224446---					
OR	B/W	s=Dn	d=--2224446---	~**00	1000	DDD1	SSEE	EEEE
		d=Dn	s=2-2224446444		1000	DDD0	SSee	eeee
	L	s=Dn	d=--2224446---		1000	DDD1	10EE	EEEE
		d=Dn	s=2-2224446446		1000	DDD0	10ee	eeee
ORI	B/W	s=Imm	d=4-4446668---	~**00	0000	0000	SSEE	EEEE
	L	s=Imm	d=6-666888A---					
ORI CCR	B	s=Imm	d=----------4	*****	0000	0000	0011	1100
ORI SR	W	s=Imm	d=----------4	*****	0000	0000	0111	1100
PEA	L		s=--2--444644-	~~~~~	0100	1000	01ee	eeee
RESET			2----------	~~~~~	0100	1110	0111	0000
ROL	B/W	count=Dn	d=2----------	~**00	1110	rrrf	SS11	1DDD
ROR		count=#1-8	d=2----------		1110	QQQf	SS01	1DDD
	L	count=Dn	d=2----------		1110	rrrf	1011	1DDD
		count=#1-8	d=2----------		1110	QQQf	1001	1DDD
Memory	W	count=1	d=--2224446---		1100	011f	11EE	EEEE
ROXL	B/W	count=Dn	d=2----------	***0*	1110	rrrf	SS11	0DDD
ROXR		count=#1-8	d=2----------		1110	QQQf	SS01	0DDD
	L	count=Dn	d=2----------		1110	rrrf	1011	0DDD
		count=#1-8	d=2----------		1110	QQQf	1001	0DDD
Memory	W	count=1	--2224446---		1110	010f	11EE	EEEE
RTE			2----------	*****	0100	1110	0111	0011
RTR			2----------	*****	0100	1110	0111	0111
RTS			2----------	~~~~~	0100	1110	0111	0101
SBCD	B	s=Dn	d=2----------	*?*?*	1000	RRR1	0000	0rrr
		s=-(An)	d=----2-------		1000	RRR1	0000	1rrr
Scc	B	cc=True	d=2-2224446---	~~~~~	0101	CCCC	11EE	EEEE
		cc=False	d=2-2224446---					
STOP			----------4	*****	0100	1110	0111	0010
SUB	B/W	s=Dn	d=2-2224446---	*****	1001	DDD1	SSEE	EEEE
		d=Dn	s=222224446444		1001	DDD0	SSee	eeee
	L	s=Dn	d=2-2224446---		1001	DDD1	10EE	EEEE
		d=Dn	s=222224446446		1001	DDD0	10ee	eeee
SUBA	W	d=An	s=222224446444	~~~~~	1001	AAA0	11ee	eeee
	L	d=An	s=222224446446		1001	AAA1	11ee	eeee
SUBI	B/W	s=Imm	d=4-4446668---	*****	0000	0100	SSEE	EEEE
	L	s=Imm	d=6-666888A---					
SUBQ	B/W	s=Imm3	d=222224446---	*****	0101	QQQ1	SSEE	EEEE
	L	s=Imm3	d=222224446---					

SUBX	B/W	s=Dn	d=2----------	*****	1001 RRR1 SS00 0rrr
		s=-(An)	d=----2-------		1001 RRR1 SS00 1rrr
	L	s=Dn	d=2----------		1001 RRR1 1000 0rrr
		s=-(An)	d=----2-------		1001 RRR1 1000 1rrr
SWAP	W		d=2----------	~**00	0100 1000 0100 0DDD
TAS	B		d=2-2224446---	~**00	0100 1010 11EE EEEE
TRAP			2----------	~~~~~	0100 1110 0100 VVVV
TRAPV			2----------	~~~~~	0100 1110 0111 0110
TST	B/W		d=2-2224446---	~**00	0100 1010 SSEE EEEE
	L		d=2-2224446---		
UNLK			-2----------	~~~~~	0100 1110 0101 1AAA

Um ein leichteres Disassemblieren zu ermöglichen, folgt jetzt eine Tabelle, die nach den höchstwertigen 4 Bit der Opcodes sortiert ist. Gegenüber obiger Tabelle sind folgende zusätzliche Buchstaben codiert:

hh = Shift Codes: %00=AS, %01=LS, %10=ROX und %11=RO.
m = 0=Register-Register, 1=Memory-Memory
oo = Operationsart: %00=Word->Register, %01=Long->Register
%10=Word->Memory und %11=Long->Memory
s = Größe des Operanden: 0=Word, 1=Long.
tt = Typ des Befehls: %00=**BTST**, %01=**BCHG**, %10=**BCLR** und %11=**BSET**.

%0000.DDD1.ttEE.EEEE - Bitverarbeitung durch Datenregister
%0000.1000.ttEE.EEEE - Statische Bitverarbeitung
%0000.rrr1.oo00.1RRR - **MOVEP**
%0000.0000.SSEE.EEEE - **ORI**
%0000.0010.SSEE.EEEE - **ANDI**
%0000.0100.SSEE.EEEE - **SUBI**
%0000.0110.SSEE.EEEE - **ADDI**
%0000.1010.SSEE.EEEE - **EORI**
%0000.1100.SSEE.EEEE - **CMPI**
%0001.RRRM.MMee.eeee - **MOVE.B**
%0010.RRRM.MMee.eeee - **MOVE.L**
%0011.RRRM.MMee.eeee - **MOVE.W**
%0100.0000.SSEE.EEEE - **NEGX**
%0100.0000.11EE.EEEE - **MOVE from SR**
%0100.0010.SSEE.EEEE - **CLR**
%0100.0100.SSEE.EEEE - **NEG**
%0100.0100.11ee.eeee - **MOVE to CCR**
%0100.0110.SSEE.EEEE - **NOT**
%0100.0110.11ee.eeee - **MOVE to SR**
%0100.1000.00EE.EEEE - **NBCD**
%0100.1000.01ee.eeee - **PEA**
%0100.1000.0100.0DDD - **SWAP**
%0100.1000.1sEE.EEEE - **MOVE.M**
%0100.1s00.1000.0DDD - **EXT**

```
%0100.1010.SSEE.EEEE  - TST
%0100.1010.11EE.EEEE  - TAS
%0100.1010.1111.1100  - ILLEGAL
%0100.1100.1see.eeee  - MOVE.M
%0100.1110.0100.VVVV  - TRAP
%0100.1110.0101.0AAA  - LINK
%0100.1110.0101.1AAA  - UNLK
%0100.1110.0110.1AAA  - MOVE to USP
%0100.1110.0110.0AAA  - MOVE from USP
%0100.1110.0111.0000  - RESET
%0100.1110.0111.0001  - NOP
%0100.1110.0111.0010  - STOP
%0100.1110.0111.0011  - RTE
%0100.1110.0111.0101  - RTS
%0100.1110.0111.0110  - TRAPV
%0100.1110.0111.0111  - RTR
%0100.1110.10EE.EEEE  - JSR
%0100.1110.11EE.EEEE  - JMP
%0100.DDD1.10ee.eeee  - CHK
%0100.AAA1.11ee.eeee  - LEA
%0101.QQQ0.SSEE.EEEE  - ADDQ
%0101.QQQ1.SSEE.EEEE  - SUBQ
%0101.CCCC.11EE.EEEE  - Scc
%0101.CCCC.1100.1DDD  - DBcc
%0110.CCCC.PPPP.PPPP  - Bcc
%0110.0001.PPPP.PPPP  - BSR
%0111.DDD0.QQQQ.QQQQ  - MOVE.Q
%1000.DDD0.SSee.eeee  - OR     <ea>,Dn
%1000.DDD1.SSEE.EEEE  - OR     Dn,<ea>
%1000.DDD0.11ee.eeee  - DIVU
%1000.DDD1.11ee.eeee  - DIVS
%1000.RRR1.0000.mrrr  - SBCD
%1001.DDD0.SSee.eeee  - SUB    <ea>,Dn
%1001.DDD1.SSEE.EEEE  - SUB    Dn,<ea>
%1001.RRR1.SS00.0rrr  - SUBX   -(Ar),-(AR)
%1001.RRR1.SS00.1rrr  - SUBX   Dr,DR
%1011.DDD0.SSee.eeee  - CMP
%1011.RRR1.SS00.1rrr  - CMPM
%1011.rrr1.SSEE.EEEE  - EOR
%1100.DDD0.SSee.eeee  - AND    <ea>,Dn
%1100.DDD1.SSEE.EEEE  - AND    Dn,<ea>
%1100.DDD0.11ee.eeee  - MULU
%1100.DDD1.11ee.eeee  - MULS
%1100.RRR1.0000.mrrr  - ABCD
%1100.DDD1.0100.0DDD  - EXG.L  Dn,Dn
%1100.AAA1.0100.1AAA  - EXG.L  An,An
%1100.DDD1.1000.1AAA  - EXG.L  Dn,An
%1101.DDD0.SSee.eeee  - ADD    <ea>,Dn
%1101.DDD1.SSEE.EEEE  - ADD    Dn,<ea>
```

```
%1101.RRR1.SS00.0rrr  - ADDX   Dn,Dn
%1101.RRR1.SS00.1rrr  - ADDX   -(An)-(An)
```

Panik-Knopf
Siehe »Reset-Taster

PARAMBLK
Dies ist eine Struktur, die dazu verwendet wird, um Parameter an eine Routine zu übergeben. Sie wird verwendet im Zusammenhang mit dem »Objekt vom Typ "G_PROGDEF". Bei der Routine handelt es sich um ein vom Programmierer definiertes »Objekt (mehr darüber bei diesem Stichwort).

Pfad
Der Atari ST hat ein »hierarchisches Filesystem. Der Weg vom »Rootdirectory zum aktuellen Directory wird Pfad genannt. Dieser beginnt im allgemeinen mit dem Laufwerkssymbol, dann folgt ein Backslash "\" und dann die weiteren Subdirectories. Alle aufgeführten Directories sind per Backslash getrennt. Ein Beispiel: "A:\SUBDIR1\SUBDIR2\". Der aktuelle Pfad eines Programms kann mit der »GEMDOS Routine $3B *Chdir()* gesetzt und mit der »GEMDOS Routine $47 *Getdir()* abgefragt werden.

PIXEL
Engl., heißt soviel wie "Picture Element". Ein P. ist das kleinste auf einem Ausgabegerät noch darstellbare Bildelement. Auf dem Atari können Sie beispielsweise im hochauflösenden Modus 640 * 400 Pixel darstellen (Horizontal * Vertikal). Bei einem Matrixdrucker ist der kleinste noch darstellbare Punkt der Druckmatrix ein P.

Programmaufbau
Jedes Programm besteht aus drei Segmenten. Das erste Segment ist das eigentliche Programmsegment und wird *TEXT-Segment* genannt. Das zweite Segment ist das *Data-Segment* und enthält alle initialisierten Daten. Um das Programm in ein EPROM bringen zu können, sollte das DATA-Segment nur Konstanten enthalten. Das dritte Segment ist das soganannte *BSS-Segment.* Dieses stellt nur Platz für alle nicht initialisierten Daten zur Verfügung. Ein eventuelles viertes Segment enthält die Symboltabelle des Programms und heißt *Symbol-Segment.*

Ein abgespeichertes Programm enthält ebenfalls diese Segmente und einige weitere Informationen über den Aufbau des Programms. Jedes gespeicherte Programm hat folgenden Aufbau:

$00	word	Enthält den Code $601A, der anzeigt, daß es sich bei diesem File um ein Programm handelt.
$02	long	Länge des Textsegments in Bytes.
$06	long	Länge des Datasegments in Bytes.
$0A	long	Länge des BSS-Segments in Bytes.
$0E	long	Länge der Symboltabelle in Bytes.
$12-$1B		Reserviert für angebliche Erweiterungen.
$1C		Hier beginnt das Textsegment.
$1C+($02)		Hier beginnt das Datasegment.

Direkt hinter dem Datasegment folgt das BSS-Segment. Dieses belegt genau ein Wort. Wichtig ist nur die Angabe der Länge dieses Segments, damit die Laderoutine *Pexec* diesen Wert in die »Base-Page übertragen kann. Diesem einen Wort folgt eventuell noch eine »Symboltabelle.

Hinter den Segmenten folgen noch Relokatierinformationen (»Relocation). Jede Adresse, die irgendwo in das Programm selbst verweist, enthält einen Wert, der genau dann richtig wäre, wenn das Programm später einmal bei Adresse Null im Speicher liegen würde. Liegt es woanders, muß dieser Wert zur Startadresse des Programms hinzuaddiert werden. Dazu muß der Laderoutine nur noch bekannt sein, welche Werte zur Startadresse hinzuaddiert werden müssen. Dazu dienen die Relokatierinformationen, die folgenden Aufbau haben. Das erste Byte ist immer Null. Das Ende der Relokatierinformationen wird ebenfalls durch ein Nullbyte angezeigt. Jeder gerade Wert zwischen 2 und 254 dient als Offset, der zur jeweils vorangegangenen relokatierten Adresse hinzuaddiert werden muß. Falls zwei zu relokatierende Werte weiter als 254 Bytes auseinanderliegen, dient der Wert 1 dazu, den Offset um 254 zu vergrößern ohne daß etwas relokatiert wird. Der erste Wert nach einer beliebig langen Folge von Einsen wird ebenfalls zu den Vielfachen von 256 noch hinzuaddiert und ist um 2 zu groß. Liegt also der erste zu relokatierende Wert genau 256 Bytes hinter dem Beginn des Text-Segments, dann beginnen die Relokatierinformationen mit den Bytes $01 und $02.

Programmbeginn

Nachdem ein Programm durch Aufruf der »GEMDOS Routine $4B *Pexec()* in den Speicher des Atari ST geladen wurde, wird der »Userstack-pointer auf das Ende des freien Speichers gesetzt. Auf diesen Stack wird zuerst die Startadresse der »Base-Page gelegt und dann noch die Startadresse des Programmes selbst. Dann wird ein "jmp (A7)" Befehl ausgeführt, der zum Start des Programms führt. Die ersten Befehle eines Programmes müssen dafür sorgen, daß der Stackpointer kurz hinter das Programm gelegt und daß der restli-

che Speicherplatz wieder freigegeben wird. Deshalb beginnt eigentlich jedes Programm mit der folgenden Befehlssequenz oder einer ähnlichen:

```
start: move.l  4(A7),A0        ;A0 zeigt nun auf die Base-Page.
       move.l  $0C(A0),D0      ;D0 := Länge des Text-Segments.
       add.l   $14(A0),D0      ;Länge des Daten-Segments addieren.
       add.l   $1C(A0),D0      ;Länge des BSS-Segments addieren.
       and.l   #$FFFFFE,D0     ;Stack auf gerade Adresse legen.
       move.l  A0,A7           ;A7 zeigt auf die Base-Page.
       adda.l  D0,A7           ;A7 ist jetzt der korrekte Stack.
       move.l  D0,-(A7)        ;Parameter Blocklänge
       move.l  A0,-(A7)        ;Parameter Startadresse.
       clr.w   -(A7)           ;Dummy Parameter.
       move.w  #$4A,-(A7)      ;Parameter Funktionsnummer.
       trap    #1              ;GEMDOS(4A) aufrufen.
       add.l   #12,A7          ;Stackpointer korrigieren.
```

Radius

Für »VDI werden Radien stets in 1/10 Grad angegeben. Ein voller Kreis geht daher von 0 bis 3600, ein rechter Winkel hat 900 Einheiten usw. Außerdem verläuft ein Kreis stets entgegen dem Uhrzeigersinn (er wird auch so gezeichnet), ein Kreisausschnitt von 0 bis 90 Grad (0 bis 900 Einheiten) läuft daher von 15 Uhr nachmittags bis 12 Uhr mittags. Siehe auch die Abbildung unten.

```
        900
         |
         |
1800 --------- 0 = 3600
         |
         |
       2700
```

Raster

Ein R. ist einfach eine sequentielle Folge von Worten (im Speicher), die ein »Bit Image definieren. Dies wird auch als »Form bezeichnet. Ein R. ähnelt einem »Sprite (oder Shape) auf anderen Computer-Systemen. Allerdings verfügt der Atari (noch) nicht über Hardware-Sprites (das ändert sich, wenn der "Bit-Blitter-Chip" erhältlich ist). Auf dem »Bildschirm ist ein R. stets ein rechteckiger Grafik-Bereich.

Um ein R. zu definieren, braucht man ein »Memory Form Definition Block (Abk.: MFDB). Diese Struktur beschreibt den Aufbau des R. Sie sieht so aus:

Offs.	Typ	Name	Erläuterung
0	long	fd_addr	Die Adresse des Rasters
4	word	fd_w	Breite des Rasters in »Pixeln
6	word	fd_h	Höhe in »Pixeln
8	word	fd_wdwidth	Breite in Wörtern
10	word	fd_stand	Flag: zeigt an, ob das Raster in Standard- oder gerätespezifischer Form ist
12	word	fd_nplanes	Anzahl der Farbebenen (bzw. Anzahl der Bits pro »Pixel)
14	word	fd_r1	Reserviert
16	word	fd_r2	Reserviert
18	word	fd_r3	Reserviert

Im folgenden die genaue Erklärung der einzelnen Strukturkomponenten:

fd_addr Dies ist die Adresse des Rasters. Wenn 'fd_addr' Null ist, so wird die Adresse des »Bildschirms stattdessen genommen und alle anderen Parameter werden mit den korrekten Werten für den »Bildschirm überschrieben.

fd_w Die Breite des Rasters wird in »Pixeln angegeben, damit ein Raster auch pixelweise definiert werden kann (bzw. bitweise). Damit muß ein Raster **nicht** an Wortgrenzen beginnen, sondern kann eine beliebige Größe besitzen.

fd_h Die Höhe des Rasters in »Pixeln. Hier gilt das Gleiche wie unter 'fd_w' gesagt.

fd_wdwidth Die Breite des Rasters in Worten. Auch wenn die Breite beliebig sein kann, so wird ein Raster immer wortweise abgespeichert. Wenn die Breite ('fd_w') nicht ein vielfaches von 16 ist, so werden die "überzähligen" Bits in einem ganzen Wort untergebracht. Wenn 'fd_w' beispielsweise zwölf beträgt, so ist 'fd_wdwidth' = 2.

fd_stand Siehe dazu die Erklärung weiter unten

fd_nplanes Dies gibt an, wieviele Bits benötigt werden, um die Farbe eines »Pixels zu definieren. In niedriger »Auflösung sind dies vier, in mittlerer zwei und in hoher ein Bit.

fd_r1 Dieser und die darauf folgenden zwei Parameter sind für uns uninteressant. **Sie müssen bei der Definition der Struktur dennoch vorhanden sein!**

Grundsätzlich werden zwei Arten der Raster unterschieden: die Standard-Form und die gerätespezifische Form. Die Standard-Form ist für **alle** GEM-Rechner gleich. Damit ist es möglich, beispielsweise »Icons und Bilder von einem Rechner auf einen anderen zu portieren, ohne sich über den Aufbau des Bildschirmspeichers den Kopf zu zerbrechen.

Die gerätespezifische Form ist vom Aufbau des Bildschirms abhängig. Der ST hat drei verschiedene gerätespezifische Formen: eine für hohe, eine für mittlere und eine für niedrige »Auflösung. Dadurch, daß es eine Standard-Form gibt, lassen sich beispielsweise »Icons von einer Auflösung zur anderen übertragen. Allerdings erscheint ein »Icon in niedriger »Auflösung als doppelt so breit und hoch wie in hoher »Auflösung, und in mittlerer ist es doppelt

so hoch. Auch von den Farben sieht man in hoher Auflösung naturgemäß nichts.

Das Standard-Format ist mit dem monochromen Format des ST identisch. Der Aufbau ist wie folgt: Jedes »Pixel wird durch ein Bit repräsentiert. Ein gesetztes Bit bedeutet ein gesetztes »Pixel. Ist das Bit Null, so ist das entsprechende »Pixel nicht gesetzt und erscheint damit als weiß. Das Raster beginnt an einer Wortgrenze, wobei das »Pixel links oben durch das Bit 15 (das am weitesten links stehende Bit) repräsentiert wird. Das folgende Wort beschreibt die nächsten 16 »Pixel der obersten Zeile usw. bis zum Ende der Rasterzeile. Dies setzt sich von Zeile zu Zeile so fort, wobei jede Zeile wieder mit einem Wort beginnt.

Dies bedeutet, daß bei hoher »Auflösung 'fd_stand' = Null ist.

Wenn 'fd_adress' auf Null gesetzt wird, so wird damit der ganze »Bildschirm angesprochen werden. Es ist aber möglich, nur einen rechteckigen Ausschnitt aus einem Raster zu wählen. Außerdem können Sie bei 'fd_adress' jederzeit auch die tatsächliche Bildschirmaddresse eintragen.

Es gibt insgesamt drei »VDI-Routinen, mit denen Raster manipuliert werden können. Diese sind:
- *»vr_trnfm()*
- *»vro_cpyfm()*
- *»vrt_cpyfm()*

Sehen Sie bitte unter dem entsprechenden Stichwort nach!

Wenn ein Raster von einer Quelle auf (oder in) ein Ziel kopiert wird, so können dabei eine ganze Reihe verschiedener Verknüpfungen von Quelle und Ziel vorgenommen werden. In der folgenden Tabelle finden Sie die verschiedenen Modi, zusammen mit einer Erklärung. Die Raster-Operationen laufen stets nach folgendem Schema ab: Resultat = Quelle [Verknüpfung] Ziel. Resultat ist das resultierende Pixel **nach** dem die Quelle (aus der kopiert wird) mit dem Ziel (auf das kopiert wird) verknüpft wurde. Die Verknüpfung geschieht bitweise.

Modus	Operation	Erläuterung
0	Resultat = 0	Löscht all Bits
1	Resultat = Ziel AND Quelle	Setzt die Bits, die **sowohl** in der Quelle als auch im Ziel gesetzt sind
2	Resultat = Quelle AND (NOT Ziel)	Wie 1, aber vorher wird das Ziel invertiert

Modus	Operation	Erläuterung
3	Resultat = Quelle	Ziel wird mit Quelle überschrieben
4	Resultat = (NOT Quelle) AND Ziel	In Quelle gesetzte Bits werden im Ziel gelöscht
5	Resultat = Ziel	Sinnlos: bewirkt nichts
6	Resultat = Quelle XOR Ziel	Exklusiv-Oder: In Quelle gesetzte Bits werden im Ziel invertiert, zweimalige Anwendung stellt Ausgangszustand wieder her
7	Resultat = Quelle OR Ziel	Oder: Ziel und Quelle werden gemischt (das Ziel scheint durch die Quelle durch, auch als Transparent-Modus bezeichnet)
8	Resultat = NOT (Quelle OR Ziel)	Wie 7, aber das Resultat wird invertiert (Transparent invers)
9	Resultat = NOT (Quelle XOR Ziel)	Wie 6, aber das Resultat wird invertiert
10	Resultat = NOT Ziel	Das Ziel wird invertiert
11	Resultat = Quelle OR (NOT Ziel)	Wie 7, aber das Ziel wird vorher invertiert
12	Resultat = NOT Quelle	Invertierte Quelle überschreibt das Ziel
13	Resultat = (NOT Quelle) OR Ziel	Wie 7, aber vorher wird die Quelle invertiert
14	Resultat = NOT (Quelle AND Ziel)	Wie 1, aber das Ganze wird invertiert
15	Resultat = 1	Alle Bits werden gesetzt

Einige der Modi (z. B. Modus 5) sind völlig sinnlos, andere lassen sich ersetzen (s. u.). Besondere Probleme ergeben sich, wenn Sie Farbe benutzen. Die oben beschriebenen Verknüpfungen werden dann auch für jedes Bit einzeln durchgeführt – bei vier Bits pro »Pixel können daraus merkwürdige Effekte resultieren. Und: die vier Bits geben keine "absolute" Farbe an, sondern dienen nur dazu, eine Farbe aus einer Palette auszuwählen. Deshalb ist zu empfehlen, auch im Farbmodus nur mit den Farben schwarz und weiß zu

arbeiten. Alles andere kann sehr komplex werden (außer, man gebraucht nur Modus 3).

Nun zu einem praktischem Beispiel. Nehmen wir einmal an, Sie wollen den Hintergrund einer »Dialogbox im Speicher zwischenspeichern, um sich das Neuzeichnen der darunterliegenden Fenster zu ersparen. Nennen wir die Routine einfach "save_rectangle". Das erste, was wir tun müssen, ist die Bereitstellung zweier Strukturen vom Typ "MFDB". Die erste bezeichnen wir als "screen", die zweite als "memory" (wir wollen ja zuerst einen Grafik-Bereich vom Schirm in den Speicher transferieren, nachher umgekehrt). Eine MFDB-Struktur besteht aus neun Komponenten, wobei sechs davon für uns interessant sind. Im folgenden bezeichnen wir die Komponente (z. B.) 'fd_adress' der Struktur "screen" als "screen.fd-_adress". C-Programmierer sind mit der Schreibweise vertraut. Alle anderen bitten wir darum, dies erst einmal so zu schlucken!

So könnte unsere Routine aussehen:

```
          save_rectangle(x, y, w, h)
word                     x, y, w, h;
```

Eingabeparameter:

word x	X-Koordinate des zu rettenden Bereichs
word y	Y-Koordinate des zu rettenden Bereichs
word w	Breite des zu rettenden Bereichs in »Pixel
word h	Höhe des zu rettenden Bereichs in »Pixel

Globale Parameter:

MFDB screen	Globale Struktur (s. o.)
MFDB memory	Globale Struktur (s. o.)
word *array	Ein Feld, das insgesamt acht Worte aufnehmen kann(&adress ist die Adresse des Feldes). Das Feld wird verwendet, um dort die Werte zu speichern, die die beiden rechteckigen Bereiche definieren.
word *buffer	Ein Puffer, der groß genug sein muß, um den zu rettenden Grafik-Bereich aufzunehmen. Um den erforderlichen Speicherplatz zu bekommen, kann beispielsweise die Routine "Malloc()" (siehe »gemdos) genommen werden.

Warum nur globale Parameter (mit Ausnahme von 'x', 'y', 'w' und 'h')? Weil wir noch eine zweite Routine brauchen, die unseren ge-

retteten Bildschirminhalt wieder zurückschreibt. Diese Routine muß wissen, wohin der Grafikbereich gehört!

```
BEGINN:
        screen.fd_adress = 0;
        memory.fd_adress = buffer;
```

Die Struktur 'screen' adressiert den »Bildschirm. Daher brauchen wir 'fd_adress' in 'screen' nur einfach auf Null zu setzen. Die Adresse, wohin der Bildschirmhintergrund gerettet werden soll, ist die Adresse von 'buffer'.

```
        array(0) = x;
        array(1) = x;
        array(2) = x + w - 1;
        array(3) = y + h - 1;
```

Somit steht in unserem Array der zu rettende Bildausschnitt: in 'array(0)' und 'array(1)' die linke obere Ecke, und in 'array(2)' und 'array(3) die rechte untere Ecke.

```
        array(4) = 0;
        array(5) = 0;
        array(6) = w - 1;
        array(7) = h - 1;
```

Der zweite Teil von 'array()' speichert die Koordinaten des Rechtecks, das im Speicher stehen soll. Da befindet sich am Speicheranfang die linke obere Ecke.

```
        memory.fd_wdwidth = w / 4 + 1;
```

Die Breite des Rechtecks in Wörtern. Für den Fall, daß die Anzahl der »Pixel nicht durch 16 teilbar ist, nehmen wir lieber ein Wort mehr.

```
        memory.fd_stand = 0;
        memory.fd_planes = 1;
```

Annahme: wir befinden uns in der hohen Auflösung. In Farbe funktioniert die Routine nicht wie hier beschrieben!

```
        v_hide_c();
```

Mauszeiger verstecken, sonst wird der mit abgespeichert.

```
        vro_cpyfm(handle, mode, &array, &screen, &memory);
word            handle, mode;
word *                          array,  screen,  memory;
```

word handle Hierbei handelt es sich um das »handle der »virtual Workstation (falls Ihnen das nichts sagt, müssen Sie unbedingt das entsprechende Stichwort nachschlagen!).

word mode Dies ist der Modus aus obiger Tabelle. In unserem Fall muß 'mode' gleich drei sein!

word *array Ist die Adresse des oben definierten Feldes

word *screen Die Adresse der MFDB-Struktur (s. o.)

word *memory Die Adresse der MFDB-Struktur (s. o.)

Damit befindet sich der angegebene Bildschirmausschnitt in unserem Puffer (= 'buffer').

```
        v_show_c();
```

Der Mauszeiger kann wieder vorgezeigt werden.

END

Damit ist die Routine zum Retten eines rechteckigen Bildschirmausschnittes zuende. Wenn Sie den Hintergrund wieder restaurieren wollen, so müssen Sie lediglich die ersten vier Werte von 'array' mit den letzten vier vertauschen. Außerdem müssen bei »*vro_cpyfm()* die beiden letzten Parameter ebenfalls ausgetauscht werden. Die Routine braucht übrigens keine Eingabeparameter! Die stehen jetzt nämlich in 'array'.

Raster-Koordinaten

Zur Positionierung von Grafik-Ausgabe wird ein Koordinaten-System benutzt. Der »VDI kennt zwei verschiedene Darstellungen: die »Normalisierten Koordinaten (Abk.: »NDC, siehe »Normalized Coordinates) und die R. Die R. sind von der jeweiligen Auflösung abhängig. Im hochauflösenden (monochromen) Modus laufen die X-Koordinaten (horizontal) von 0 bis 639, die Y-Koordinaten (vertikal) von 0 bis 399. Der Punkt 0, 0 liegt dabei in der linken oberen, der Punkt 639, 399 in der rechten unteren Ecke. Jedem so ansprechbaren Punkt entspricht genau ein »Pixel. In der mittleren Auflösung laufen die Koordinaten von 0, 0 bis 639, 199 und in der niedrigen Auflösung von 0, 0 bis 319, 199. Drucker wiederum bieten im allgemeinen eine ganz andere Auflösung, ebenso Plotter

usw., d. h. die verwendeten Koordinatensysteme sind gerätespezifisch. Ein Programm, das diese R. benutzt, kann nur auf dem Gerät laufen, für das es geschrieben wurde. Um dieses Problem zu lösen, kennt »VDI noch die "»Normalized Coordinates" (Abk. NC, siehe unter dem entsprechenden Stichwort). Da »VDI aber für jedes Koordinatenpaar dies erst in die gerätespezifische Form umrechnen muß, sind Programme, die NC benutzen, langsamer. Außerdem kann es durch die Umrechnungen zu Rundungsfehlern kommen, so daß die Ausgabe nicht dem gewünschten Bild entspricht. Daher benutzen die meisten Programme lieber gleich Raster-Koordinaten. Außerdem muß bei der Verwendung von Raster-Koordinaten »GDOS installiert worden sein. Die gewünschte Koordinaten-Darstellung wird beim Aufruf von »*v_opnvwk()* angegeben.

Raubkopie

Siehe »Justiz

Rechteckliste

Bei der R. handelt es sich um eine Liste, die »AES im Zusammenhang mit »Fenstern aufbaut. Dies sind die rechteckigen Bereiche, die mit bestimmten Fenstern assoziert sind, und in die eine »Anwendung hineinzeichnen kann, ohne andere Fenster zu zerstören. Falls Sie in einem »Fenster zeichnen wollen, sollten Sie erst die Rechteckliste befragen, ob Sie damit nichts kaputt machen. Siehe »*wind_get()* und insbesondere »Window Update.

Relocation

Engl.: Verschiebung. Wenn die »GEMDOS Routine $4B *Pexec()* ein Programm laden soll, sucht sie den Beginn des freien Bereiches der »TPA. Dorthin wird das Programm geladen. Der Beginn dieses Bereiches hängt jedoch davon ab, wieviele Programme sich bereits in der TPA befinden. Daher ist beim Schreiben eines Programmes noch nicht bekannt, wo das Programm liegen wird. Aus diesem Grund werden alle Programme so erzeugt, daß sie ab Adresse Null richtig liegen würden. Wenn sie dann an eine beliebige (gerade!) andere Adresse gebracht werden, müssen alle Adressen, die als Operanden bei einem Befehl auftauchen, für diese neue Startadresse umgerechnet werden. Zu diesem Zweck werden hinter jedem Programm noch sogenannte Relokatierinformationen abgelegt (»Programmaufbau).

Request

Engl., wörtlich "Gesuch". »VDI kennt zwei verschiedene Methoden, eine Eingabefunktion zu gestalten: den "Request"-Modus und den "»Sample"-Modus. Im ersteren Modus kehrt die Routine zurück, wenn der Benutzer eine Eingabe gemacht hat, im zweiten Fall wird nur

der Status des Eingabegeräts abgefragt (also ob eine Eingabe vorliegt).

Reset-Taster

Der R.-Taster befindet sich and er Rückseite vom Atari und löst einen Warmstart des Betriebsystems aus. Wenn in der »Systemvariable *resvalid* ($424) $31415926 steht, wird über den Inhalt der »Systemvariable *resvector* ($42A) gesprungen.

Resource-File

Hierbei handelt es sich um eine Datei mit der »Extension ".RSC". Diese enthält die Daten, die den Aufbau von »Objekten (»Dialogboxen, »Warnboxen, »Drop-Down-Menüs etc.) beschreiben. Erstellt werden sie meistens mit einem Resoure Construction Program. Verwaltet werden die Datenstrukturen dann mit Funktionen aus der »Resource Library. Siehe auch »Objekt und insbesondere »Dialogbox (unter letzterem Stichwort finden Sie Hinweise für die Verwendung von Resourcen).

Resoure Library

Für die meisten »GEM-Programme existiert neben der eigentlichen Programmdatei noch eine weitere Datei mit einer »Extension ".RSC". In dieser Datei befinden sich die Datenstrukturen, die zur Erzeugung von »Dialogboxen, zur Darstellung von »Drop-Down-Menüs, »Icons und allen anderen »Objekten dienen.

Der Vorteil dieser Trennung von Daten und Programmen liegt darin, daß Änderungen in diesen Daten (z. B. Übersetzung von Menütexten in eine andere Landessprache) gemacht werden können, ohne das Programm zu ändern. Ferner ist es möglich, die »RSC-Files von einem Rechner zum anderen ohne große Änderungen zu portieren. Ein weiterer Vorteil liegt darin, daß die Erstellung der Resourcen durch ein eigenes Programm auf visuelle Art geschieht, nicht durch abstrakte Programmiertätigkeit.

Die Schritte zur Verwendung der ".RSC"-Files sind folgende:

1.) Das entsprechende RSC-File wird mithilfe eines »Resource Construction Programs erstellt. Mit diesem Programm lassen sich nach dem Baukastenprinzip z. B. »Dialogboxen erstellen. Die Datenstrukturen dieser »Objekte werden vom Programm erzeugt. Jedes Objekt erhält einen (symbolischen) Namen, der im Programm als Konstante verwendet werden kann.

2.) Mittels »*rsrc_load()* wird das RSC-File vom Programm nachgeladen.

3.) Mit der Routine *rsrc_gaddr()* können nun die Adressen der »Objekte geholt werden.

4.) Damit steht alles zur Verfügung, um ein »Objekt zu zeichnen (»*objc_draw()*), ein »Drop-Down-Menü (»*menu_bar()*) anzuzeigen oder eine »Dialogbox (»*objc_draw()* und »*form_do()*) zu erzeugen: diese Routinen z. B. erwarten als ein Parameter die Adresse des »Objekts.

In der Resource Library finden Sie folgende Routinen:

-»*rsrc_load()* Lädt das RSC-File in den Speicher
-»*rsrc_gaddr()* Bestimmt die Adresse von Objekten
-»*rsrc_free()* Gibt den Speicherplatz, den das RSC-File belegt, wieder frei.
-»*rsrc_obfix()* Wandelt Zeichenkoordinaten in Pixelkoordinaten um.
-»*rsrc_saddr()* Speichert den Index einer Datenstruktur.

Unter dem Stichwort »Dialogbox finden Sie Anwendungshinweise für die Resourcen.

Rollbalken

Beim R. handelt es sich um einen Fensterbestandteil, mit dem es möglich ist, ein Fenster horizontal oder vertikal zu scrollen. Siehe auch »Window.

Rollpfeil

Beim R. handelt es sich um einen Fensterbestandteil, mit dem es möglich ist, ein Fenster horizontal oder vertikal zu scrollen. Siehe auch »Window.

Root-Directory

Jeder physikalische Massenspeicher, der einen wahlfreien Zugriff auf die gespeicherten Daten erlaubt, enthält genau eine Root-Directory. Damit ist die Wurzel des Directorybaumes eines »hierarchischen Filesystems gemeint.

RS-232

siehe »V.24 Schnittstelle

RSC-File

Siehe »Resource File und »Resource Library

Rsrc_free

Zur »Resource Library gehörende Routine.

```
$6F = 111        re_freturn = rsrc_free()
word             re_freturn;
```

Mit dieser Routine wird der mittels »*rsrc_load()* belegte Speicherplatz wieder freigegeben.

Eingabeparameter: keine

```
control(0)  =  111
control(1)  =    0
control(3)  =    0
```

Ausgabeparameter:

```
word re_freturn     0 = Ein Fehler ist geschehen, n = alles Ok.

control(2)  =    1
control(4)  =    0

int_out(0)  =  re_freturn
```

Rsrc_gaddr

Zur »Resource Library gehörende Routine.

```
$70 = 112        re_greturn = rsrc_gaddr(re_gtype, re_gindex,
word                                     re_gtype, re_gindex;
                                         &re_gaddr)
word *                                    re_gaddr;
```

Mittels dieser Routine kann man (nach Laden des RSC-Files durch »*rsrc_load()*) die Adressen von Datenstrukturen (Objekte z. B.) herausfinden. Wie Sie die Datenstrukturen erstellen können, können Sie dem Abschnitt über »Dialogboxen entnehmen. Dort finden Sie auch ein Beispiel.

Was es mit einem »Resource-File auf sich hat, können Sie unter dem entsprechenden Stichwort nachschlagen.

Eingabeparameter:

word re_gtype — Der Typ der Datenstruktur, deren Adresse zurückgegeben werden soll:

0 = Baum
1 = OBJECT
2 = TEDINFO
3 = ICONBLK
4 = BITBLK
5 = string
6 = imagedata
7 = obspec
8 = te_ptext
9 = te_ptmplt
10 = te_pvalid
11 = ib_pmask
12 = ib_pdata
13 = ib_ptext
14 = bi_pdata
15 = ad_frstr
16 = ad_frimg

Was diese einzelnen Objekttypen zu bedeuten haben, können Sie unter dem Stichwort »Objekte nachlesen.

word *re_gaddr — Zeiger auf eine Variable, in die die Routine die Adresse des gewünschten »Objekts hineinschreibt.

control(0) = 112
control(1) = 2
control(3) = 0

int_in(0) = re_gtype
int_in(1) = re_gindex

Ausgabeparameter:

word re_greturn — 0 = Ein Fehler ist vorgekommen, n = alles Ok.

long re_gaddr — Adresse des gewünschten Objekts.

control(2) = 1
control(4) = 1

int_out(0) = re_greturn

addr_out(0) = re_gaddr

Rsrc_load

Zur »Resource Library gehörende Routine.

```
$6E = 110        re_lreturn = rsrc_load(re_lpfname)
byte *                                  re_lpfname;
```

Diese Routine stellt für ein Resource-File Speicher zur Verfügung und lädt die entsprechende Datei . Weitere Aufrufe von »*rsrc_gaddr()* setzen diesen Schritt voraus. Dies ist notwendig, falls das Programm nicht alle Resourcen schon in seiner eigenen Datenstruktur integriert hat (was ohnehin nicht empfehlenswert ist).

Die Resource-Files haben nach allgemein üblicher Konvention denselben Namen wie das Programm und die Erweiterung ".RSC". Das ist aber nicht zwingend vorgeschrieben.

Sollte das Programm sein RSC-File nicht laden können, muß es den Programmlauf abbrechen. Die Adresse, an die das RSC-File geladen wurde, ist in dem Global-Array zu finden (siehe unter »AES).

Eingabeparameter:

byte *re_lpfname	Zeiger auf einen String, der den Namen des zu ladenden RSC-Files (optional mit Laufwerksangabe und/oder Pfadnamen) enthält. Wird das Laufwerk nicht angegeben, so wird das Aktuelle genommen. Gibt man den Pfad nicht an, so wird das aktuelle Directory benutzt. Der Name des RSC-Files darf die üblichen "Joker" (= »Wildcards) enthalten, wie etwa "*" und "?".

```
control(0)  =  110
control(1)  =    0
control(3)  =    1

addr_in(0)  =  re_lpfname
```

Ausgabeparameter:

word re_lreturn	0 = Ein Fehler ist geschehen (z. B. Datei nicht gefunden, weil die Datei sich nicht im selben Directory befand), n = alles Ok.

```
control(2)  =    1
control(4)  =    0

int_out(0)  =  re_lreturn
```

Rsrc_obfix

Zur »Resource Library gehörende Routine.

```
$72 = 114        re_oresvd = rsrc_obfix(re_otree, re_oobject)
word             re_oresvd,                       re_oobject;
word *                                  re_otree;
```

Mit dieser Routine ist es möglich, die Koordinaten eines »Objekts umzurechnen – von Zeichenkoordinaten nach Pixelkoordinaten. Keine Routine, die auf dem Atari unentbehrlich ist.

Eingabeparameter:

```
word *re_otree      Adresse des »Objekts Index des zu konvertie-
                    renden »Objekts

word re_oobject     Indexnummer des »Objekts

control(0)  =  114
control(1)  =    1
control(3)  =    1

addr_in(0)  =  re_otree
```

Ausgabeparameter:

```
word re_oresvd      Immer gleich 1

control(2)  =  1
control(4)  =  0

int_out(0)  =  re_oresvd
```

Rsrc_saddr

Zur »Resource Library gehörende Routine.

```
$71 = 113        re_sreturn = rsrc_saddr(re_stype, re_sindex,
word             re_sreturn,             re_stype, re_sindex;
                                         &re_saddr)
word *                                   re_saddr;
```

Hiermit kann man den Zeiger auf ein »Objekts ändern, und ihn beispielsweise auf ein neu geschaffenes »Objekt richten.

Eingabeparameter:

word re_stype	Der Typ des »Objekts, eine Zahl von NULL bis 16 (siehe 're_gtype' aus der Beschreibung von »*rsrc_gaddr()*.
word re_sindex	Die Indexnummer des Objekts, dessen Adresse geändert werden soll.
word *re_saddr	Diese Adresse wird in der durch 're_stype' und 're_sindex' angegebenen Datenstruktur gespeichert.

```
control(0)  =  113
control(1)  =    2
control(3)  =    1

int_in(0)   =  re_stype
int_in(1)   =  re_sindex

addr_in(0)  =  re_saddr
```

Ausgabeparameter:

word re_sreturn	0 = Ein Fehler ist geschehen, n = alles Ok.

```
control(2)  =    1
control(4)  =    0

int_out(0   = re_sreturn
```

Sample
siehe »Request

Scancode
Es gibt verschiedene Funktionen, die zur Abfrage der Tastatur verwendet werden können (*Bconin()* vom »BIOS, *Cconin()*, *Crawio()*, *Crawcin()* und *Cnecin()* von den »GEMDOS-Funktionen und »*evnt_-keybd()* und »*evnt_multi()* von den »AES-Routinen). Der »Tastaturprozessor des Atari liefert aber nicht den »ASCII-Code der gedrückten Taste, sondern den sog. Scancode. Dieser entspricht im allgemeinen der Anordnung der Tasten. Beim Atari allerdings wird der IBM-kompatible Scancode vom Tastaturprozessor zurückgegeben.

Im Anhang D sehen Sie die Tastatur des Atari im Bild. Neben der gewohnten Tastaturbeschriftung stehen auf den Tasten noch die Scancodes. Ruft man beispielsweise die Funktion *Bconin()* auf, so liefert diese in dem unteren Wort des zurückgegebenen Langwortes den »ASCII-Code der gedrückten Taste und in dem unteren Byte des höherwertigen Wortes den Scancode (siehe Beschreibung dieser Routine unter dem Stichwort »BIOS).

Einige Tasten liefern auch keinen »ASCII-Code. Dazu gehören die Funktionstasten, aber auch z. B. <Help>- und die <Undo>-Tasten, die Cursor-Tasten usw. Will man abfragen, ob eine dieser Tasten gedrückt wurde, so muß man den Scancode testen. Mit Hilfe des Scancodes kann man auch herausfinden, ob eine Zifferntaste der Hauptastatur oder des Ziffernblocks betätigt wurde. Mit Ausnahme der Funktionstasten wird der Scancode durch Drücken der Shift-Tasten nicht beeinflußt.

Um herauszufinden, welche Taste welchen Scancode zurückliefert, sehen Sie sich bitte Anhang D an. Dort finden Sie auch eine komplette Tastaturtabelle.

Schnittstellen
Der Atari St ist mit S. recht gut ausgestattet. Die Pinbelegungen der Schnittstellen finden Sie im Anhang.
- »Centronics-Port
- »V.24-Schnittstelle
- Zwei »MIDI Anschlüsse
- Zwei Anschlüsse für Joysticks o.ä. (»Tastaturprozessor)
- Ein Erweiterungsport für zusätzliche 128 kByte Programm (»ROM-Cartridge)
- Eine »DMA Schnittstelle zum Anschluß von bis zu 8 (?) externen Geräten.
- Nicht zu vergessen ein Monitoranschluss

Scrap Library

Die S. enthält eine Reihe von Routinen, um Daten zwischen verschiedenen Programmen austauschen zu können. Die Daten sind standardisiert und werden auf Diskette gespeichert, wobei das Programm dafür verantwortlich ist, die korrekten Datenformate einzuhalten.

Zwei verschieden Operationen können ausgeführt werden: es ist möglich, Daten auszuschneiden (die Daten werden dann aus der Quelle gelöscht) oder sie zu kopieren (die Daten bleiben in der Quelle erhalten). Beim Lesen der Daten aus dem so gebildeten "Klemmbrett" werden diese stets kopiert, so daß die Daten beliebig oft dupliziert werden können. Es existiert immer nur **ein** Klemmbrett zur Zeit (pro »Directory), eine neue Schneide- oder Kopieraktion löscht die alten Daten des Klemmbretts.

Folgende Routinen existieren:
- »*scrp_read()* Liest Daten aus dem zuletzt abgespeicherten Klemmbrett
- »*scrp_write()* Schreibt Daten in das aktuelle Scrap-Directory

Folgende Standardtypen für Daten werden von »GEM unterstützt (die Daten werden aufgrund der »Extension unterschieden):

- »ASCII-Text (.TXT)
- Tabellen-Kalkulationsdaten (.DIF)
- Metafile - das sind Dateien, die Grafikdaten nach »GEM-Standard enthalten
- »Bit-Images - Bilder, die Bitweise abgespeichert wurden

Der Datei-Name lautet stets "SCRAP.*" (wobei .* für eine der obigen »Extensions steht).

Scrap_read

Zur »Scrap-Library gehörende Routine.

```
$50 = 80        sc_rreturn = scrp_read(sc_rpscrap)
word            sc_rreturn;
byte *                                  sc_rpscrap;
```

Mit dieser Routine wird das SCRAP-Directory gelesen. Das entsprechende Directory wird einfach an die angegebene Stelle in den Speicher (Puffer) kopiert. Es muß darauf geachtet werden, daß der Puffer eine ausreichende Länge hat! Dazu sollte das Programm vorher die Länge des Directory in Erfahrung bringen.

Eingabeparameter:

```
byte *sc_rpscrap    Die Adresse des Puffers, in den das Scrap-Di-
                    rectory geladen wird

control(0)   =  80
control(1)   =   0
control(3)   =   1

addr_in(0)   =  sc_rpscrap
```

Ausgabeparameter:

```
word sc_rreturn     0 = Ein Fehler ist geschehen (z. B. Datei
                    nicht gefunden), n = alles in Ordnung

control(2)   =   1
control(4)   =   0

int_out(0)   =  sc_rreturn
```

Scrap_write

Zur »Scrap-Library gehörende Routine.

```
$51 = 81        sc_wreturn = scrp_write(sc_wpscrap)
word            sc_wreturn;
byte *                                  sc_wpscrap;
```

Diese Routine schreibt das Scrap-Directory auf Diskette (siehe auch »*scrp_read()*). Eine eventuell vorher vorhandenes Directory wird überschrieben.

Eingabeparameter:

```
byte *sc_wpscrap    Zeiger auf den Puffer, von dem das neue
                    Scrap-Directory auf das "Klemmbrett" kopiert
                    wird

control(0)   =  81
control(1)   =   0
control(3)   =   1

addr_in(0)   =  sc_wpscrap
```

Ausgabeparameter:

```
word sc_wreturn     0 = Ein Fehler ist geschehen (z. B. Datei
                    nicht gefunden), n = alles in Ordnung
```

```
control(2)   =   1
control(4)   =   0

int_out(0)   =   sc_wreturn
```

Screen

Engl., zu deutsch: »Bildschirm. Siehe »Bildschirmspeicher und Bildschirmsteuerung

Screen Manager

Dieser Bestandteil von »AES ist für die Überwachung der »Maus (genauer: des »Mauszeigers) sowie der »Drop-Down-Menüs verantwortlich. Der S. sorgt beispielsweise dafür, daß sich ein »Drop-Down-Menü entfaltet, und er meldet auch die Wahl des Benutzers an das Programm zurück (über eine »Message). Bei »GEM-Programmen ist der S. stets präsent.

Sektoren

S. befinden sich auf Diskette. In ihnen werden die zuspeichernden Infomartionen untergebracht. Ein S. ist beim Atari normalerweise 512 Bytes groß.

Semaphore

Eine S. ist ein softwaremäßiger Schalter (meist ein gesetztes oder gelöschtes Bit), der den Zugriff auf bestimmte Teile des Rechners regelt. Im allgemeinen werden S.n nur in »Multitask Umgebungen benötigt. Dort wird z.B. dem Drucker eine S. zugeordnet, die den Zugriff zum Drucker regelt. Ist die Semaphore frei, wird sie belegt und die Task beginnt mit dem Drucken. Kommt nun eine andere Task an die Reihe und möchte etwas drucken, so muß sie warten, bis die Semaphore von der ersten Task wieder freigegeben wurde. Ansonsten würde es recht schwer lesbare Listings geben.

Shel_envrn

Zur »Shell Library gehörende Funktion

```
$7D = 125      sh_eresvd = shel_envrn(sh_epvalue, sh_eparm)
word           sh_eresvd;
long *                                 sh_epvalue;
byte *                                             sh_eparm;
```

Mit dieser Routine kann man den sog. »Environment String herausfinden (siehe »GEMDOS, *Pexec()*). Dieser String kann zur Übergabe

von Parametern an Programme dienen. Dies wird aber nur von wenigen Programmen auch tatsächlich genutzt.

Eingabeparameter:

long *sh_epvalue Addresse einer Variablen für Rückgabeparameter(s. u.)

byte *sh_eparm Nach dem hier angegebenen Parameter-String soll die Routine suchen. Dies kann z. B. "PATH=" sein.

```
control(0)  =  125
control(1)  =    0
control(2)  =    1
control(3)  =    3
control(4)  =    0

addr_in(0)  =  sh_epvalue
addr_in(1)  =  sh_eparm
```

Ausgabeparameter:

word sh_eresvd Ist immer gleich Eins

long epvalue Dies ist ein Zeiger. Er zeigt auf das Byte, das direkt hinter dem gesuchten Parameter-String steht.

```
int_out(0)  =  sh_eresvd
```

Shel_find

Zur »Shell Library gehörende Funktion

```
$79 = 121        sf_wreturn = shel_find(sh_f_pbuff)
word             sf_wreturn;
byte *                                  sh_f_pbuff;
```

Diese Routine gibt von einem gegebenen Dateinamen den kompletten Suchpfad wieder. Findet die Funktion eine Datei nicht auf dem angegebenen Laufwerk, so sucht sie außerdem noch auf dem Laufwerk "A:\".

Eingabeparameter:

byte *sh_f_pbuff Dies ist ein Zeiger auf einen String, in dem die gesuchte Datei steht (z. B. "Test.dat") Nach der Rückkehr (s. u.) steht der komplette

Suchpfad in diesem String, der mindestens 80 Byte lang sein muß.

```
control(0)  =  124
control(1)  =    0
control(2)  =    1
control(3)  =    1
control(4)  =    0

addr_in(0)  =  sh_f_pbuff
```

Ausgabeparameter:

word sh_freturn	Gibt an, ob die Datei gefunden wurde oder nicht: 0 = Datei nicht gefunden (oder anderer Fehler) 1 = Datei gefunden
byte *sh_f_pbuff	Falls 'sh_freturn' gleich Eins ist, so steht in dem Puffer der komplette Suchpfad, im obigen Beispiel könnte dort "A:\test.dat" stehen.

```
int_out(0)  =  sh_freturn
```

Shel_read

Zur »Shell Library gehörende Funktion

```
$78 = 120       sh_rreturn = shel_read(sh_rpcmd, sh_rptail)
word            sh_rreturn;
byte *                                  sh_rpcmd, sh_rptail;
```

Mit dieser Routine kann ein Programm herausfinden, wie es aufgerufen wurde, mit welchem Kommando und welche Parameter an das Programm übergeben wurden. Das Kommando besteht im allgemeinen aus dem Namen des aufgerufenen Programms. Parameter werden vom »Desktop aus nur an Programme mit der »Extension ".TTP" übergeben.

Eingabeparameter:

byte *sh_rpcmd	Addresse des Puffers, in den das Kommando zurückgegeben werden soll (s. u.). Der Puffer sollte mindestens 16 Byte lang sein.
byte *sh_rptail	Addresse des Puffers, in das die Aufruf-Parameter eingetragen werden sollen (s. u.). Länge: mindestens 80 Zeichen (Byte)

```
control(0)  =  120
control(1)  =    0
control(2)  =    1
control(3)  =    2
control(4)  =    0

addr_in(0)  =  sh_rpcmd
addr_in(1)  =  sh_rptail
```

Ausgabeparameter:

word sh_rreturn	0 = Ein Fehler ist aufgetreten 1 = Alles in Ordnung
byte *sh_rpcmd	In dem Puffer steht das Kommando, mit dem das Programm aufgerufen wurde
byte *sh_rptail	Enthält die Parameter, die das aufrufende Programm übergeben hat

```
int_out(0)  =  sh_rreturn
```

Shel_write

Zur »Shell Library gehörende Funktion

```
$79 = 121        sh_wreturn = shel_write(sh_wdoex, sh_wisgr,
word             sh_wreturn,             sh_wdoex, sh_wisgr;
                                         sh_wiscr, sh_wpcmd,
word                                     sh_wiscr;
byte *                                               sh_wpcmd;
                                         sh_wptail)
byte *                                   sh_wptail;
```

Mit dieser Routine können Sie von einem laufenden Programm aus ein anderes aufrufen. Das aufgerufene Programm startet entweder sofort oder erst nach Beendigung des gerade laufenden.

Nicht jede Kombination der Parameter 'sh_wdoex', 'sh_wisgr' und 'sh_wiscr' ist sinnvoll. Falls Sie ein Programm sofort starten ('sh_wiscr = 1), so endet das aufgerufene Programm mit einem Absturz. Einwandfrei funktioniert diese Routine nur, wenn 'sh_wiscr' gleich Null ist.

Falls Sie ein Programm sofort starten möchten, empfehlen wir daher die »Gemdos-Routine *Pexec()*.

Eingabeparameter:

word sh_wdoex — Hiermit wird angegeben, ob Ihr nach Beendigung des gerade laufenden Programms das System neu gebootet werden soll, oder ob ein anderes Programm gestartet werden soll:
0 = System neu »Booten (Programme im AUTO-Ordner werden aber nicht ausgeführt!)
1 = Andere »Anwendung starten

word sh_wisgr — Falls 'sh_wdoex' = 1:
Benutzt das aufgerufene Programm Grafik?
0 = Nein, kein Grafik-Programm (»TOS-Anwendung, »Bildschirm wird gelöscht und »Maus wird versteckt)
1 = Ja, das Programm macht von Grafik Gebrauch (»GEM-Anwendung)

word sh_wiscr — Ist das nächste Programm eine »GEM-Programm?
0 = Kein »GEM-Programm, Programm wird sofort gestartet
1 = Ja, es ist eine »GEM-Anwendung, die nach Beendigung des laufenden Programms gestartet wird

byte *sh_wpcmd — Zeiger auf einen Puffer, der den Namen des zu startenden Programms enthält.

byte *sh_wptail — Zeiger auf die Kommando-Parameter, die an das nächste Programm übergeben werden sollen.
Beispiel: Die folgende Zeile steht irgendwo in Ihrem C-Programm:

```
shel_write(1,1,1,"B:\WORDPLUS.PRG",
           "TEST.DOC");
```

Dann wird nach Beendigung Ihres Programms das Programm "WORDPLUS.PRG" von Laufwerk B: geladen und gestartet, wobei das Programm angewiesen wird, eine Datei namens "TEST.DOC" zu laden bzw. zu bearbeiten.

```
control(0)  =  121
control(1)  =    3
control(2)  =    1
control(3)  =    2
control(4)  =    0

int_in(0)   =  sh_wdoex
int_in(1)   =  sh_wisgr
int_in(2)   =  sh_wiscr
```

addr_in(0) = sh_wpcmd
addr_in(1) = sh_wptail

Ausgabeparameter:

word sh_wreturn Fehlermeldung:
0 = Ein Fehler ist geschehen
1 = Alles in Ordnung

int_out(0) = sh_wreturn

Shell

Engl.: Schale. Dieser Begriff in Bezug auf Computer entstammt der UNIX-Welt. Damit ist ein »Kommandointerpreter gemeint, der das Betriebssystem wie eine Schale umhüllt. Er wandelt die Eingaben des Benutzers um in Aufrufe von Betriebssystemroutinen und steuert auch deren Ausführung. Die beim Atari ST benutzte S. ist grafischer Art und wird »GEM genannt. Diese Art von S. wertet weniger textuelle Eingaben aus, sondern interpretiert die mit der Maus durchgeführten Aktionen und wandelt diese in Aufrufe des Betriebssystems um.

Shell Library

Die S. wird vom »Desktop dazu benutzt, um andere Programme aufzurufen und zu starten. Außerdem können aufgerufene Programme mit den zugehörigen Routinen herausfinden, welche Kommando-Parameter an sie übergeben worden sind.

Da der »Desktop eine »Anwendung wie jede andere auch ist, können Sie für Ihre Programme natürlich die S. ebenfalls benutzen, um von Ihrem Programm aus bei Bedarf andere Programme aufzurufen. Dazu können Sie natürlich auch die »GEMDOS-Funktion *Pexec()* verwenden.

Die S. besteht aus folgenden Routinen:

- »*shel_envrn()* Sucht nach dem Vorkommen des sog. "Environment"-Parameters
- »*shel_find()* Sucht nach einer Datei im aktuellen »Directory
- »*shel_read()* Holt das Kommando, mit dem eine »Anwendung aufgerufen wurde
- »*shel_write()* Ruft eine andere »Anwendung auf

Sprite

Unter einem S. versteht man ein grafisches Gebilde, welches frei auf dem »Bildschirm bewegt werden kann. Meistens werden S. bei

Spielen verwandt, und einige Rechner unterstützen dies sogar mit ihrer Hardware. Der Atari kennt allerdings nur Software-Sprites. Der »Mauszeiger beispielsweise ist ein solcher Sprite. Dieser wird mit den »Line-A-Befehlen bewegt.

Siehe auch »Mauszeiger und »Icon.

Soundchip

Im Atari ist ein S. vom Typ YM-2149 eingesetzt. Dieses IC der Firma Yamaha wurde ursprünglich entwickelt, um vielfältige Funktionen in Videospielen zu übernehmen. Es besitzt 2 bidirektionale 8bit breite Ports und drei voneinander unabhängig programmierbare Oszillatoren. Jeder Oszillator hat einen eigenen Ausgang, diese sind im Atari jedoch zusammengeschaltet. Der S. besitzt insgesamt 16 Register mit einer Größe von jeweils 8 bit.

Auf diese Register wird folgendermaßen zugegriffen: Zuerst wird in Adresse $FF8800 (8bit) die Nummer des gewünschten Registers [0;15] geschrieben. Dann wird in Adresse $FF8802 (8bit) der gewünschte Wert geschrieben. Um das angewählte Register auszulesen, muß von Adresse $FF8800 gelesen werden. Die Nummer des angewählten Registers bleibt solange erhalten, bis sie überschrieben wird. Dies geschieht zumindest alle 5 ms, weil in der Timer-Interrupt Routine (»Interruptstruktur) überprüft wird, ob die Diskette gewechselt wurde. Da man nie weiß, wann dieser Interupt stattfindet, sollte ein Beschreiben eines Registers mit dem MOVEP-Befehl (»Opcodes) und ein Lesen durch direkt aufeinanderfolgende Befehle erfolgen. Geht es nur darum, Geräusche zu erzeugen, sollte der Programmierer die »XBIOS Funktion 32 *dosound()* benutzen.

Es folgt nun die Beschreibung der Register, wobei die Nummer diejenige ist, die in $FF8800 geschrieben werden muß, um das Register anzusprechen.

REG 0+1: Alle Bits des Registers 0 plus die unteren 4 Bits des Registers 1 bilden einen 12 Bit Wert, der die Frequenz des Oszillators A bestimmt. Je kleiner dieser Wert ist, desto höher ist die Frequenz.

REG 2+3: Wie Register 0+1, jedoch für Oszillator B.

REG 4+5: Wie Register 0+1, jedoch für Oszillator C.

REG 6: Die Bits 0-4 dieses Registers bestimmen die Frequenz des Rauschgenerators. Je kleiner dieser Wert ist, desto höher ist die Frequenz.

REG 7: Dieses Register steuert den Soundchip:
Bit 0 = Schaltet Oszillator A ein (=0)
Bit 1 = Schaltet Oszillator B ein (=0)
Bit 2 = Schaltet Oszillator C ein (=0)
Bit 3 = Mischt das Rauschen mit Oszillator A (=0)
Bit 4 = Mischt das Rauschen mit Oszillator B (=0)
Bit 5 = Mischt das Rauschen mit Oszillator C (=0)
Bit 6 = Steuert Port A als Eingang (=0)
Bit 7 = Steuert Port B als Eingang (=0)

REG 8: Die Bits 0-3 dieses Registers bestimmen die Lautstärke des vom Oszillator A erzeugten Tones. Je höher der Wert ist, desto lauter ist der Ton. Ist Bit 4 gesetzt, dann wird das Signal von Oszillator A durch die Hüllkurve bestimmt, und die Bits 0-3 werden ignoriert. Die Bits 5-7 haben keine Funktion.

REG 9: Wie Register 8, jedoch für Oszillator B.

REG 10: Wie Register 8, jedoch für Oszillator C.

REG 11+12 Diese beiden Register bestimmen gemeinsam die Periodendauer der Hüllkurve.

REG 13: Die Bits 0-3 legen die Art der Hüllkurve fest. Dabei handelt es sich jedoch nicht um eine wirkliche Hüllkurve vom Typ ADSR (Attack, Decay, Sustain, Release), sondern um eine grobe Signalformsteuerung.

REG 14: Die Bits dieses Registers entsprechen den Signalen am Port A des Chips (Pins 14-21). Im Atari ST werden diese Pins folgendermaßen benutzt:
Bit 0 = Side-Select Signal für Floppylaufwerke
Bit 1 = Drive-Select Signal für Laufwerk A
Bit 2 = Drive-Select Signal für Laufwerk B
Bit 3 = RTS Signal für die »V.24-Schnittstelle
Bit 4 = DTR Signal für die »V.24-Schnittstelle
Bit 5 = Strobe Signal am »Centronics-Port
Bit 6 = Monitor-Buchse Pin 3
Bit 7 = Nicht benutzt.

REG 15: Die Bits dieses Registers entsprechen den Signalen am Port B des Chips (Pins 6-13) und sind direkt, also ohne zwischengeschaltete Treiber, mit dem »Centronics-Port verbunden!!

Speicherkonfiguration

An Adresse $FF8001 (ungerade !) befindet sich ein acht Bit langes Register der Atari-MMU, das für die S. verantwortlich ist. Relevant sind jedoch nur die Bits Null bis Drei. Diese Bits können folgende Werte annehmen:

Wert	Bank 0	Bank 1	Wert	Bank 0	Bank 1
%0000	128 kByte	128 kByte	%0001	128 kByte	512 kByte
%0010	128 kByte	2 MByte	%0011	reserviert	
%0100	512 kByte	128 kByte	%0101	512 kByte	512 kByte
%0110	512 kByte	2 MByte	%0111	reserviert	
%1000	2 MByte	128 kByte	%1001	2 MByte	512 kByte
%1010	2 MByte	2 MByte	%1011	reserviert	
%11xx	alle weiteren sind reserviert				

Dabei gilt, daß 128 kByte aus 16 Chips zu je 64k*1 Bit bestehen. 512 kByte bestehen aus 16 512k*1bit Chips und 2 MegaByte setzen sich aus 16 1M*1bit Chips zusammen.

Sprungweite

Die »CPU 68000 besitzt Branchbefehle, die relativ zum aktuellen Programmzähler mit einer 8 oder 16 Bit großen Sprungweite verzweigen. Dieser »Offset ist in Zweierkomplementform angegeben. Hat der 8 Bit »Offset in einem Branchbefehl den Wert Null, dann holt sich der Prozessor das folgende Wort aus dem Speicher und benutzt es als 16 Bit Sprungweite.

Statusregister

Ein S. ist ein Register in einem hochintegrierten Schaltkreis, das den Zustand (Status) dieses Schaltkreises angibt. Jeder Prozessor und viele »I/O-Chips besitzen solche Register. Von den im Atari ST verwendeten Chips besitzen der »MC 68000, die »ACIA's 6850, der Controller »FDC 1772 und der Multifunktionschip »MFP 68901 jeweils ein oder mehrere S.

Subdirectory

Als S. bezeichnet man im »hierarchischen Filesystem ein Directory, das weitere Files und/oder Subdirectories enthält. Das Directory in der obersten Ebene bezeichnet man im Gegensatz zum Subdirectory als »Rootdirectory ('\'). Ein S. hat beim Atari ST einen bis zu 11 Zeichen langen Namen, genau wie ein normales File.

Supervisormode

Der im Atari ST benutzte Prozessor »MC 68000 kann in zwei verschiedenen Modi arbeiten. Der normale Modus, in dem alle Benutzerprogramme ablaufen sollten, ist der »Usermode. Der S. ist der Modus, in dem Routinen des Betriebssystems abgearbeitet werden. In welchem Modus sich die CPU befindet, wird durch Bit 13 ihres Statusregisters angezeigt. Ein gesetztes Bit zeigt den S. an. Dieses Bit wird gesetzt, wenn die CPU eine »Exception ausführt. Sie geht dann automatisch in den S. über.

Der S. zeichnet sich gegenüber dem »Usermode dadurch aus, daß es einige Befehle gibt, die nur in diesem Modus erlaubt sind. Außerdem gibt es Speicherstellen, wie z.B. die »I/O-Chips, die ebenfalls nur in diesem Modus angesprochen werden dürfen.

Symboltabelle

Das »Filesystem des Atari ST kann S.n verwalten. Eine S. enthält alle im Programm verwendeten Symbole wie Namen von Routinen oder globalen Variablen und befindet sich im Programmfile hinter dem Programmsegment (»Programmaufbau). Jeder Eintrag in der S. besteht aus genau sieben Worten und ist folgendermaßen aufgebaut:

Wort 0-3 Enthalten den bis zu acht Zeichen langen Namen des Symbols. Ist der Name kürzer, wird er mit Nullbytes aufgefüllt.

Wort 4 Enthält den Typ des Symbols. Dieser ist bitweise kodiert und wird durch die Bits 15-8 angegeben. Es gibt folgende Werte:
Bit 0-7 Nicht benutzt.
Bit 8 = Symbol des BSS-Segments (»Programmaufbau)
Bit 9 = Symbol des Text-Segments (»Programmaufbau)
Bit 10 = Symbol des Data-Segments (»Programmaufbau)
Bit 11 = Extern definiert.
Bit 12 = Wert ist eine Registernummer.
Bit 13 = Global deklarierter Wert.
Bit 14 = Normale Variable.
Bit 15 = Konstante.

Wort 5+6 Enthalten den bis zu 32 Bit langen Wert dieses Symbols.

Eine solche S. ist außerordentlich hilfreich beim Entwickeln von Programmen. Leider wird sie zur Zeit noch von zu wenigen Compilern unterstützt, denn eine solche Tabelle muß zunächst einmal erstellt und an das File angehängt werden. Dann wird auch noch ein Debugger benötigt, der beim Disassemblieren und bei Speicherzugriffen Symbole unterstützt.

Systemvariablen

In der folgenden Auflistung der S. sind nur die Variablen aufgeführt, bei denen von Atari zugesichert wurde, daß sie auch in zukünftigen Versionen des Betriebssystems nicht geändert werden:

$400 etv-timr (long)
Zeiger auf die Routine, die periodische Aufgaben des »GEM erledigt. Sie wird durch den System-Timer Interrupt aufgerufen.

$404 etv-crtc (long)
Zeiger auf eine Routine, die versucht, Diskettenfehler zu korrigieren.

$408 etv_term (long)
Über diesen Zeiger wird eine Routine angesprungen, die ein Programm beendet.

$40C etv_xtra (5long)
Hier ist Platz für 5 weitere Zeiger, die zur Zeit jedoch nicht genutzt werden.

$420 memvalid (long)
enthält die magische Zahl $752019F3, die zusammen mit *memval2* anzeigt, daß der Wert in *memcntrl* gültig ist.

$424 memcntrl (byte)
enthält das Byte, das nach einem Reset in den Memory-Controller des Atari ST geschrieben werden soll (»Speicherkonfiguration).

$426 resvalid (long)
enthält diese Adresse nach einem Reset den Wert $31415926 (läßt sich gut für PI im BCD-Format gebrauchen), dann wird über den Vektor in *resvector* ($42A) gesprungen.

$42A resvectr (long)
Dieser Vektor zeigt auf eine Routine, die nach einem Reset angesprungen wird. Er wird aber nur benutzt, wenn *resvalid* ($426) die magische Zahl enthält.

$42E phys_top (long)
Enthält einen Zeiger auf das erste nicht mehr benutzbare Byte des freien RAM-Speichers.

$432 membot (long)
Enthält einen Zeiger auf den Beginn des frei verfügbaren RAM-Speichers.

$436 memtop (long)
Enthält einen Zeiger auf das letzte Byte des RAM's. Dieser Wert ist um genau 1 kleiner als der von *phys_top* ($42E).

$43A memval2 (long)
Enthält die magische Zahl $237698AA, die zusammen mit *memvalid* ($420), anzeigt, daß der Kontrollwert in *memcntrl* ($424) gültig ist.

$43E flock (word)
Befindet sich hier ein Wert ungleich Null, so werden Zugriffe des Betriebssystems auf den Atari »DMA-Chip während der »V_Blank Routine verhindert. Dies ist besonders für Festplattenzugriffe wichtig.

$440 seekrate (word)
Die Bits Null und Eins enthalten die »Seekrate für beide Floppylaufwerke:

Wert	0	1	2	3
Zeit	6 ms	12 ms	2 ms	3 ms

$442 timr_ms (word)
Hier steht die Zeit, die zwischen zwei Timeraufrufen vergeht. Der übliche Wert ist 20 ms ($14=20). Dies entspricht 50 Hz. Dieser Wert wird von der »BIOS-Routine Nummer 6 *Tickcal* zurückgeliefert.

$444 fverify (word)
Wenn diese Variable einen Wert ungleich Null enthält, werden alle Schreibzugriffe auf ein Laufwerk verifiziert. Nach einem Reset wird defaultmäßig Verify eingeschaltet.

$446 bootdev (word)
Enthält die Nummer des Devices, von dem gebootet wurde.

$448 palmode (word)
Ist diese Variable ungleich Null, dann befindet sich das System im PAL-Modus (50 Hz Video). Ist sie Null, befindet sich das System im NTSC-Modus (60 Hz Video).

$44A defshift (word)
Hier muß der Wert für die gewünschte neue Bildschirmauflösung eingetragen werden, wenn diese verändert werden soll.

$44C sshiftmd (word)
Enthält einen Duplikatwert des Hardwareregisters für die Bildschirmauflösung. Es sind folgende Werte möglich:

0 320*200 Punkte in 16 Farben (A0)
1 640*200 Punkte in 4 Farben (A1)
2 640*400 Punkte in 2 Farben (A2 = monochrom)

$44E v_basad (long)
Zeiger auf den Beginn des »Bildschirmspeichers.

$452 vblsem (word)
»Semaphore, mit der die Ausführung der »V-Blank Routinen gesteuert wird. Um diese Routinen auszuführen, muß diese Variable den Wert 1 haben.

$454 nvbls (word)
Enthält die Anzahl der Zeiger, auf die der Zeiger *_vblqueu* ($456) zeigt. Der Defaultwert ist 8.

$456 _vblqueu (long)
Diese Variable enthält einen Zeiger auf einen Vektor von Zeigern, die die Startadressen der »V-Blank Routinen enthalten.

$45A colorptr (long)
Zeiger auf 16 Worte, die beim nächsten »V-Blank in die Hardware Farbregister übertragen werden sollen. Nach der Übertragung wird dieser Zeiger auf Null gesetzt. Damit wird eine weitere Übertragung vermieden.

$45E screenpt (long)
Wenn dieser Zeiger ungleich Null ist, wird er beim nächsten »V-Blank als neue Startadresse des Bildschirmspeichers benutzt.

$462 _vbclock (long)
Diese Variable zählt die »V-Blank Interrupts mit.

$466 _frclock (long)
Diese Variable zählt die ausgeführten »V-Blank Routinen mit, also all die, die nicht durch die Semaphore *vblsem* ($452) verhindert wurden.

$46A hdv_init (long)
Zeiger auf die Initialisierungsroutine für Festplatten. Wenn unbenutzt, enthält diese Variable den Wert Null.

$46E swv_vec (long)
Zeiger auf die Routine, die beim einem Wechsel vom monochromen auf den Farbmonitor oder umgekehrt aufgerufen wird.

$472 hdv_bpb (long)
Zeiger auf eine Routine, die den »BIOS Parameter Block für eine Festplatte zurückliefert. Zum Aufruf wird die »BIOS Routine 7 *Getbpb* benutzt. Wenn unbenutzt, enthält diese Variable den Wert Null.

$476 hdv_rw (long)
Zeiger auf eine Routine zum Lesen von und zum Schreiben auf eine Festplatte. Diese Routine wird über die »BIOS Routine 4 *Rwabs* aufgerufen. Wenn unbenutzt, enthält diese Variable den Wert Null.

$47A hdv_boot (long)
Zeigt auf die Routine zum Booten von einer Festplatte. Null, wenn unbenutzt.

$47E hdv_medi (long)
Zeigt auf eine Routine, die den Media Change Wert für eine Festplatte zurückliefert. Zum Aufruf dient die »BIOS Routine 9 *Mediach*. Null, wenn unbenutzt.

$482 cmdload (word)
Wenn diese Variable ungleich Null ist, versucht das Betriebssystem nach einem Reset ein Programm namens "COMMAND.PRG" vom Bootdevice zu laden. Diese Variable kann z.B. von einem »Bootsektor ungleich Null gesetzt werden.

$484 conterm (byte)
Diese Variable enthält Kontrollbits für die »Console. Die Bits haben folgende Bedeutung:
Bit 0 - Es piept, wenn das Zeichen ^G (ASCII 7) ausgegeben wird.
Bit 1 - Ermöglicht die Tastenwiederholung. Bit 2
Bit 2 - Es klickt, wenn eine Taste gedrückt wird. Bit 3
Bit 3 - Bei »Tastaturabfragen wird *Kbshift* (»BIOS 11) mitgeliefert.

$48E themd (4*long)
»Memory descriptor. Wird durch die »BIOS Routine 0 *getmpb* ausgefüllt.

$49E md (long)
Platz für weitere »Memory Descriptoren.

$4A2 savptr (long)
Zeiger auf einen Speicherbereich, in den die Registerinhalte nach einem »BIOS-Aufruf gerettet werden.

$4A6 nflops (word)
Enthält die Anzahl der angeschlossenen Floppy-Disk Laufwerke.

$4A8 con_stat (long)
Enthält einen Zeiger für Bildschirmausgaben, der je nach auszugebendem Wert auf die entsprechende Routine gesetzt wird (z.B. für »Escape-Sequenzen).

$4AC save_row (word)
Zwischenspeicher für die aktuelle Cursorzeile beim Positionieren mit ESC 'Y'.

$4AE sav_cont (long)
Zeiger auf einen Speicherbereich, in den nach einer »Exception die Registerinhalte gerettet werden.

$4B2 bufl (2*long)
Enthält zwei Zeiger auf Listheader für einige GEMDOS Routinen. Der erste zeigt auf den Header für Datensektoren, der zweite auf den Header für »FAT und »Directory. Jeder dieser Header hat folgenden Aufbau:

(long)	0 -	Zeiger auf den jeweils anderen Header
(word)	4 -	enthält die Laufwerksnummer oder den Wert -1.
(word)	6 -	Puffertyp
(word)	8 -	Aktuelle Recordnummer in diesem Puffer.
(word)	10 -	Ist dieses Flag ungleich Null, wurde der Pufferinhalt verändert.
(long)	12 -	Enthält einen Zeiger auf einen »Drive Media Descriptor.
(long)	16 -	Zeiger auf diesen Header.

$4BA hz_200 (long)
Enthält einen Zähler für den 200 Hz Systemtakt. Dieser wird durch 4 geteilt, um den 50 Hz Videotakt zu erhalten.

$4BE the_env (4*byte)
Enthält den Default »Environment String (vier Nullbytes).

$4C2 drvbits (long)
Enthält einen 32 Bit Wert, der anzeigt, welche Laufwerke angeschlossen sind. Bit 0 steht für Laufwerk A, Bit 1 für B, usw. Man erhält diesen Wert durch Aufruf der »BIOS Routine 10 *Drvmap*.

$4C6 dskbufp (long)
Zeiger auf einen 1024 Byte großen Zwischenpuffer für Laufwerkszugriffe. Dieser Puffer wird auch von einigen »VDI-Routinen benutzt und steht deshalb für Interruptroutinen nicht zur Verfügung!

$4CA autopth (long)
Zeiger auf einen Defaultpath. Zeigt normalerweise auf Null, wird also nicht benutzt.

$4CE vbl_lst (8*long)
Enthält die Zeiger auf die acht Routinen, die bei einem »V-Blank Interrupt ausgeführt werden sollen.

$4EE prt_cnt (word)
Diese Variable wird zu -1 initialisiert. Gleichzeitiges Drücken der Tasten Alternate und Help inkrementiert diesen Wert. Ein Wert von Null führt zum Ausdruck einer Bildschirm Hardcopy. Nochmaliges Drücken von Alternate und Help dekrementiert diesen Wert wieder und führt zum Abbruch der Hardcopy.

$4F2 sysbase (long)
Enthält einen Zeiger auf den Beginn des Betriebssystems.

$4F6 shell_p (long)
Zeiger auf einen »Shell spezifischen Inhalt.

$4FA end_os (long)
Zeiger auf den Beginn der »TPA. Dort endet der RAM Bereich, der vom Betriebssystem beansprucht wird.

$4FE exec_os (long)
Zeiger auf den Beginn des »AES. Dorthin wird nach der Initialisierung des »BIOS gesprungen, um das »AES zu initialisieren und die »GEM Benutzeroberfläche aufzubauen.

$502 dump_vec (long)
Zeiger auf eine Routine zum Ausdrucken einer Hardcopy. Die »XBIOS Routine 20 *Scrdump* benutzt diesen Vektor.

$506 prt_stat (long)
Zeiger auf eine Routine, die den Druckerstatus bei einer Hardcopy meldet.

$50A prt_vctr (long)
Zeiger auf die Routine für die Druckerausgabe bei Hardcopies.

$50E aux_stat (long)
Zeiger auf eine Routine, die den Status der »V24-Schnittstelle liefert.

$512 aux_vctr (long)
Zeiger auf die Ausgaberoutine für die »V24-Schnittstelle.

Task-ID

Jedes ablaufende Programm (Application) im Atari hat eine Identifikations-Nummer ('ap_id'). Diese Nummer (wird bei der Anmeldung von »*appl_init()* vergeben) benötigen Programme, um untereinander Informationen auszutauschen. Folgende Routinen machen Gebrauch von der Task-ID:

- »*appl_init()*
- »*appl_exit()*
- »*appl_find()*
- »*appl_read()*
- »*appl_tplay()*
- »*appl_trecord()*
- »*appl_write()*
- »*menu_register()*

Tastaturabfrage

Das Betriebssystem des Atari ST stellt mehrere Möglichkeiten zur T. zur Verfügung. Zunächst kann der Status der Tastatur mit der »BIOS Routine $01 *Bconstat(2)* oder mit der »GEMDOS Routine $0B *Constat()* abgefragt werden. Um Zeichen einzulesen, stehen die »BIOS Routine $02 *Bconin(2)* und die »GEMDOS Routinen $01 *Cconin()*, $07 *Crawcin()*, $08 *Cnecin()* und $0A *Cconrs(buf_ptr)* zur Verfügung. Das »AES bietet zur T. die Routinen $14 *evnt_keydb()*, $15 *evnt_button()* und $32 *form_do()* an. Bitte sehen Sie unter den genannten Stichworten nach.

Tastaturprozessor

Der Atari ST besitzt einen zweiten Prozessor vom Typ 6301. Dabei handelt es sich um einen Single-Chip Mikroprozessor, der über 4 kByte ROM, 128 Byte RAM, vier 8 Bit breite I/O-Ports und eine serielle Schnittstelle verfügt. Über die I/O-Ports werden die Tastatur und Maus bzw. Joysticks abgefragt. Über die serielle Schnittstelle läuft die Kommunikation mit dem Hauptrechner ab. Diese Schnittstelle wird auf der Hauptplatine von einem der beiden »ACIA's 6850 bedient.

Damit der T. richtig arbeitet, muß er programmiert werden, bzw. müssen ihm nach einem Reset Befehle übermittelt werden, die seine Arbeitsweise festlegen. Diese Befehle werden mittels der »XBIOS Routine $19 *Ikbdws(bytes, pointer)* an den T. gesendet. Die folgende Tabelle ist nach Befehlsbytes sortiert. Jeder Befehl, mit dem man bestimmte Stati des T.s setzen kann, kann auch zum Abfragen des aktuellen Status dienen. Dazu muß im Befehl einfach Bit 7 gesetzt werden, und es werden natürlich dann keine weiteren Parameter mit gesendet. Der T. liefert dann ein Paket mit einem

Header von $F6 zurück. Diesem Header folgen dann genau die Parameter, die auch beim Setzen des Status benutzt werden.

$07

Es muß noch ein Parameterbyte folgen, das so interpretiert wird:

Bit 0 Ist dieses Bit gesetzt, wird beim Drücken einer der beiden Maustasten die absolute Position der Maus gemeldet. Bit 2 muß gelöscht sein.

Bit 1 Ist dieses Bit gesetzt, wird die absolute Mausposition beim Loslassen einer Maustaste gemeldet.

Bit 2 Dieses Bit darf nur dann gesetzt sein, wenn Bit 0 und 1 gelöscht sind. Ist es gesetzt, dann wird ein Drücken einer Maustaste wie eine normale Taste gemeldet. Die linke Taste liefert den Code $74, die rechte den Code $75.

Bit 3-7 Diese Bits müssen alle gelöscht sein.

$08

Ab jetzt wird immer die relative Mausposition als Differenz zur vorigen gemeldet. Ist diese Differenz in einem Byte nicht mehr darstellbar, weil die Maus zu schnell bewegt wurde, werden automatisch zwei oder auch mehr Pakete erzeugt. Diese Pakete werden vom T. automatisch gesendet, wenn eine bestimmte Anzahl an Impulsen von der Maus überschritten wurde. Diese Schwelle wird mit dem Befehl $0B festgelegt. Das vom T. gesendete Paket hat folgenden Aufbau:

Byte 0 $F8-$FB. Kennzeichnet ein relatives Mauspositionspaket. Die Bits 0 und 1 geben den Zustand der beiden Maustasten an.

Byte 1 Differenz zur alten X-Position im Zweierkomplement.

Byte 2 Differenz zur alten Y-Position im Zweierkomplement.

$09

Ab jetzt wird auf Anfrage immer die absolute Mausposition gemeldet. Gleichzeitig werden die maximalen Werte für die X- und Y-Position festgelegt. Mauspositionen, die über diesen Maxima oder unter dem Nullpunkt liegen, werden bereits vom T. ignoriert. Die ersten zwei Byte hinter dem Befehl $09 legen die maximale X-Koordinate und das zweite Wort legt die maximale Y-Koordinate fest.

$0A

Ab jetzt werden statt den Koordinaten der Maus immer die Scancodes der Cursortasten geliefert. Das bedeutet, nachdem die Maus eine bestimmte Anzahl an Taktimpulsen ausgelöst hat, z.B. indem sie nach rechts bewegt wurde, wird der Scancode für Cursor rechts an den Atari ST gesendet. Das erste Byte hinter dem Befehl $0A legt die Anzahl an Impulsen fest, die von der Maus ausgelöst werden müssen, bevor der Code für Cursor rechts oder links gesendet wird. Das zweite Byte legt dasselbe für die Y-Richtung fest.

$0B
Mit diesem Befehl wird die Schwelle festgelegt, nach der der T. eine Positionsänderung der Maus an den Atari ST melden soll. Dies gilt natürlich nur, sofern vorher mit dem Befehl $08 der relative Modus eingeschaltet wurde. Das erste Byte hinter diesem Befehl gibt die Schwelle für die X-Richtung an und das zweite die für die Y-Richtung. Diese Schwelle wird als Koordinatendifferenz angegeben.

$0C
Mit diesem Befehl wird die Mausbewegung skaliert. Das erste Byte gibt an, nach welcher Anzahl an Mausimpulsen der Koordinatenzähler für die X-Position verändert werden muß. Das zweite Byte tut dies für die Y-Position.

$0D
Abfrage der absoluten Mausposition. Als Antwort sendet der T. ein Paket mit folgendem Aufbau:

Byte 0 Header ($F7) zur Kennzeichnung eines absoluten Mauspakets
Byte 1 Tastenstatus. Dieser ist Bitweise codiert:
Bit 0 - 1= Rechte Maustaste wurde seit der letzten Abfrage gedrückt.
Bit 1 - 1= Rechte Taste wurde nicht gedrückt.
Bit 2 - 1= Linke Maustaste wurde seit der letzten Abfrage gedrückt.
Bit 3 - 1= Linke Taste wirde nicht gedrückt.
Byte 2+3 Absolute X-Position der Maus.
Byte 4+5 Absolute Y-Position der Maus.

Aus dem etwas merkwürdig aussehenden Tastenstatus kann erkannt werden, daß eine Taste zwischen zwei Abfragen mehrfach gedrückt wurde, wenn sowohl Bit Null als auch Bit 1 (bzw. Bit 2 und 3) gesetzt sind.

$0E
Dieser Befehl dient zum Setzen der aktuellen Position der Maus. Das Paket muß folgenden Aufbau haben:

Byte 0 Muß Null sein. Es dient als Füllbyte, um einen wortweisen Aufbau des Pakets zu erreichen.
Byte 1+2 Absolute X-Position.
Byte 3+4 Absolute Y-Position.

$0F
Legt den Ursprung der Y-Koordinate nach unten. Damit wird ein *normales* Koordinatensystem erzeugt.

$10
Legt den Ursprung der Y-Koordinate nach oben. Wir haben dann ein gespiegeltes Koordinatensystem, das aber dem physikalischen Aufbau des »Bildschirmspeichers entspricht.

$11
Dieser Befehl sorgt dafür, daß der T. wieder Daten zum Atari ST sendet (siehe Befehl $13). Stattdessen kann jedoch auch jeder andere Befehl (mit Ausnahme von $13 natürlich) für diesen Zweck benutzt werden.

$12
Schaltet die Maus aus, sofern sie sich nicht im Modus $0A befindet, in dem statt Mauspositionen nur Cursorcodes gesendet werden. Die Maus wird durch jeden der Befehle $08, $09 und auch $0A wieder eingeschaltet.

$13
Schaltet die gesamte Datenübertragung zum Atari ST aus. Mausbewegungen und Tastendrücke werden in dem kleinen Puffer (<64 Byte) des T. zwischengespeichert. Die Übertragung wird mit dem Befehl $11 oder mit jedem anderen Befehl wieder eingeschaltet.

$14
Sorgt dafür, daß jede Bewegung eines angeschlossenen Joysticks automatisch gemeldet wird. Die vom T. gesendeten Pakete haben folgendes Format:

Byte 0 Header ($FE) für Joystick 0 und $FF für Joystick 1.
Byte 1 Die Bits 0-3 geben die Stellung des Joysticks an. Jedem Schalter entspricht ein Bit. Die Bits 4-6 werden nicht benutzt und Bit 7 gibt an, ob der Feuerknopf gedrückt ist (1) oder nicht (0).

$15
Hebt den Befehl $14 wieder auf. Um jetzt den oder die Joysticks abzufragen, muß entweder ein Paket mit $16 angefordert werden, oder man schaltet das automatische Melden mit $14 wieder ein.

$16
Mit diesem Befehl werden Joystickpakete angefordert. Der T. sendet für jeden Joystick ein Paket, das den unter $14 beschriebenen Aufbau besitzt.

$17
Erzwingt eine Dauermeldung der Joysticks. Das Byte hinter dem Befehl gibt die Zeit in 1/100 sec an, die zwischen zwei Paketen vergehen soll. Ein vom T. gesendetes Paket hat folgenden Aufbau:

Byte 0 Ein gesetztes Bit 0 zeigt an, daß der Feuerknopf von Joystick Null gedrückt ist. Bit 1 tut dies für den Feuerknopf von Joystick 1.
Byte 1 Die Bits 0 bis 3 zeigen die Stellung des Joysticks 0, und die Bits 4 bis 7 die des Joysticks 1 an.

$18
Erzwingt eine Dauermeldung des Feuerknopfes von Joystick 1. Diese wird in einem Byte gepackt, wobei Bit 7 als erstes belegt wird, dann Bit 6 usw. bis zum Bit 0. Die Joystickabfragen erfolgen in konstanten Zeitabständen. Außer diesen Meldungen sendet der T. solange keine weiteren Pakete, bis er einen anderen Befehl empfangen hat.

$19
Statt den Daten für Joystick 0 werden die Scancodes der Cursortasten an den Rechner gemeldet. Dieser Befehl benötigt sechs Parameter, die Zeitintervalle angeben. Alle diese Intervalle werden in 1/10 sec gemesssen. Wird der Joystick Null eine bestimmte Zeit in eine Richtung gehalten, so geschieht folgendes: Bis zum Ablauf der Zeit T_0 werden in Abständen von T_1 Cursorcodes an den Rechner gesendet. Nach Ablauf des Intervalls T_0 werden in Abständen von T_2 Cursorcodes an den Rechner geschickt. Alle diese Intervalle werden für die X- und die Y-Richtung getrennt angegeben. Es gilt folgender Aufbau:

Byte 0	T_0 (x)	Muß entweder Null oder größer als T_1 sein.
Byte 1	T_0 (y)	
Byte 2	T_1 (x)	Sendeabstand bevor T_0 abgelaufen ist.
Byte 3	T_1 (y)	
Byte 4	T_2 (x)	Sendeabstand nach Ablauf von T_0.
Byte 5	T_2 (y)	

$1A
Mit diesem Befehl werden alle Joystickmodi ausgeschaltet.

$1B
Hiermit wird die Uhrzeit gesetzt, die sich der T. intern hält. Diese wird im packed-BCD Format gesendet. Der T. ignoriert aber alle Halbbytes, die keine gültige BCD-Ziffer enthalten. Gültige BCD-Ziffern sind die Werte von %0000=0 bis %1001=9. Die Werte von %1010=$A bis %1111=$F sind ungültig und werden komplett ignoriert. Dadurch ist es möglich, daß nur Teile der Uhrzeit verändert werden, wie z.B. das Datum. Ein Uhrzeitpaket muß folgendem Aufbau genügen:

Byte 0	Jahreszahl im BCD-Format (z.B. $86 für 1986).
Byte 1	Nummer des Monats (z.B. $12 für Dezember).
Byte 2	Nummer des Tages (z.B. $31 für den 31. Tag des Monats).
Byte 3	Stunde im 24h Format (z.B. $23 für 23 Uhr abends).
Byte 4	Minuten (z.B. $59 für die 59. Minute dieser Stunde).
Byte 5	Sekunden (z.B. $59 für die 59. Sekunde...)

$1C
Fordert die aktuelle Uhrzeit vom T. an. Der T. sendet ein aus 7 Bytes bestehendes Paket, das den Header $FC hat und dem sechs

weitere Bytes folgen, die die Uhrzeit wie eben beschrieben enthalten.

$1D bis $1F werden nicht benutzt.

$20
Ein Paket, das mit diesem Code beginnt, lädt Daten direkt in den Speicher des T.s. Die zwei direkt folgenden Bytes geben die Startadresse an, ab der die Daten liegen sollen. Allerdings besitzt der T. nur im Bereich von $80 bis $FF ein klein wenig RAM-Speicher. Das nächste Byte gibt die Anzahl an Datenbytes an, die sofort dahinter folgen müssen.

$21
Mit diesem Befehl kann der Speicher des T.s ausgelesen werden. Es werden immer 6 Byte lange Pakete vom T. zurückgeliefert. Die beiden Bytes direkt hinter dem Befehl $21 geben die Startadresse an, ab der der Speicher ausgelesen werden soll. Der T. sendet dann ein Paket mit folgendem Aufbau:

Byte 0 $F6 dieser Header zeigt an, daß es sich um ein Statuspaket des T.s handelt.
Byte 1 $20 dieser Subheader zeigt an, daß es sich um einen angeforderten Speicherinhalt handelt.
Byte 2-7 enthalten die sechs Bytes aus dem Speicher des T.s.

$22
Ruft eine bestimmte Routine des T.s auf, deren Startadresse in den zwei Bytes stehen müssen, die diesem Befehl direkt folgen.

Tedinfo
Bezeichnung für eine Struktur, die Informationen über vom Benutzer editierbaren Text enthält (Text-Editier-Information). Die Struktur ist unter dem Stichwort »Objekt erläutert, der Umgang mit editierbaren Textfeldern ist bei »Dialogbox erklärt.

TOS
Abk. für "Tramiel Operating System". TOS ist ein eingetragenes Warenzeichen der Atari Corp.

TOS ist der eigentliche Name des Betriebssystems für den Atari (Jack Tramiel ist der Name des Besitzers von Atari). Zum TOS im engeren Sinn gehören das »BIOS, das »XBIOS, das »GEMDOS und der »Line-A-Emulator. Als TOS-Anwendungen bezeichnet man Programme, die keine »GEM-Routinen verwenden. In der Regel sind das reine Textprogramme.

TPA

Abk. für -Transient Program Area-. Die TPA ist der Speicherbereich im Atari, der vom Gemdos für Programme zur Verfügung gestellt wird und nicht vom »TOS benutzt wird. Der Begin der TPA steht in der »Systemvariable *_membot* ($432) und das Ende steht in der »Systemvariable *_memtop* ($436).

Tracebit

Hierbei handelt es sich um ein Bit im Statusregister der CPU »MC 68000. Wird dieses Bit gesetzt, dann springt die CPU nach jedem ausgeführten Befehl über den »Exception Vektor Nummer 9 in Adresse $000024 in eine vom Benutzer anzugebende Routine. Diese Routine gibt dann z.B. die Inhalte aller Register aus und erlaubt Eingaben des Benutzers. Diese Funktion der CPU stellt eine sehr leicht handhabbare Möglichkeit zum Debuggen vom Programmen dar, denn es ist einfach ein Programm in Einzelschritten zu bearbeiten.

Trap

Engl.: Falltür. Mit Traps werden bestimmte »Exceptions bezeichnet, die vom Programmierer zum Aufruf von Routinen des Betriebssystems oder für eigene Zwecke verwendet werden können. Diese Traps sind normale Befehle (»Opcodes) der CPU MC68000.

Usermode

Dies ist einer der beiden Modi, in dem die CPU »MC68000 des Atari ST arbeiten kann. Dieser Modus ist eingeschaltet, wenn Bit 13 des Statusregisters der CPU gelöscht ist. Alle Anwendungsprogramme sollten in diesem Modus arbeiten, denn in diesem Modus ist die Benutzung einiger privilegierter Befehle nicht erlaubt und auch der Zugriff auf bestimmte Speicherbereiche ist untersagt. Außerdem darf vom Statusregister der CPU nur das untere Byte benutzt werden. Wenn die CPU im Usermode einen privilegierten Befehl ausführen soll, tritt eine »Exception auf und die CPU springt über den Vektor 2 in Adresse $000020 in eine Routine, die diese Privilegverletzung bearbeitet. Dies führt zu sichereren Programmen (»Supervisormode).

User-Stackpointer

Die CPU »MC68000 besitzt zwei Register A7. Diese Register werden üblicherweise als Stackpointer benutzt. Welches der beiden Register gerade aktiv ist, wird durch das Supervisorbit (Bit 13) im Statusregister der CPU festgelegt. Ein im »Usermode laufendes Programm arbeitet mit dem User-Stackpointer. Der System-Stackpointer zeigt beim Atari ST im allgemeinen in einen Bereich in den unteren 4 kByte des Speichers. Im Supervisormode gibt es einen speziellen Befehl (move USP), mit dem man Zugriff auf den User-Stackpointer hat.

V.24-Schnittstelle

Die V. des Atari ST stellt auf physikalischer Ebene im wesentlichen alle Leitungen zur Verfügung. Leider werden einige dieser Leitungen von den Routinen des Betriebssystems nicht unterstützt. Der Eingabestatus dieser Schnittstelle kann mit der »GEMDOS Routine $12 *Cauxis()* oder mit der »BIOS Routine $01 *Bconstat(1)* abgefragt werden. Der Ausgabestatus wird mit der »GEMDOS Routine $13 *Cauxos()* oder mit der »BIOS Routine $08 *Bcostat(1)* ermittelt. Zum Einlesen von Zeichen stehen die »GEMDOS Routine $03 *Cauxin()* und die »BIOS Routine $02 *Bconin(1)* zur Verfügung. Zeichen können mittels der »GEMDOS Routine $04 *Cauxout(char)* und mit der »BIOS Routine $03 *Bconout(1,char)* ausgegeben werden. Die Parameter dieser Schnittstelle können mit der »XBIOS Routine $0F *Rsconf()* gesetzt werden.

VBL

Abk. für -Vertical Blank Interrupt-. Der VBL-Handler steht auf der Adresse, die in der Speicherzelle $70 steht. Der Handler erhöht die »Systemvariable *_frclock* ($466) und testet die »Systemvariable *vblsem* ($452). Wenn sie ungleich Null ist, dann werden keine weiteren Routinen vom VBL ausgeführt, ansonsten wird die »Systemvariable *_vblclock* ($462) erhöht. Weiterhin werden auch alle Register auf dem Stack gesichert.

Danach wird Monochromdetected überprüft. Falls der Rechner in der höchsten Auflösungsstufe gearbeitet hat, wird sicherheitshalber auf niedrige Auflösung umgeschaltet, wenn kein monochromdetected Signal mehr existiert, da einige Monitore von dem Synchronisationssignal für die hohe Auflösung zerstört werden können.

Daraufhin wird die Cursor-Blinkroutine aufgerufen.

Falls die »Systemvariable *colorptr* ($45a) ungleich null ist, wird eine neue Farbpalette eingestellt, worauf *colorptr* zeigt. Danach wird *colorptr* wieder auf Null gesetzt.

Falls die »Systemvariable *screenpt* ($45e) ungleich null ist, wird der Bildschirm auf die Adresse verlegt, die in *screenpt* steht, und *screenpt* wird auf null gesetzt.

Danach werden die Routinen ausgeführt, die in den VBL-Slots liegen. An dieser Stelle können Sie eigene Routinen in den VBL einbinden. Ihre Routine sollte mit einem **RTS** enden und sie dürfen alle Register verändern bis auf den User Stack Pointer (»USP).

Mit folgender Assemblerroutine (muß im Supervisormode ausgeführt werden) können sie eigene Routinen in den VBL einbinden:

```
install: move.w  $454,d0     * Anzahl der Slots
         lsl.w   #2,d0       * mit vier multiplizieren
         move.l  $456,a0     * lade Adresse des ersten Slots
         clr.w   d1          * lösche d1
find:    tst.l   0(a0,d1)    * teste Slot, ob er frei ist
         beq     freier      * wenn ja, verzweige
         addq.w  #4,d1       * sonst erhöhe d1 um 4
         cmp.w   d0,d1       * und teste, ob letzter Slot erreicht
         bne     find        * wenn nicht, suche weiter
         rts                 * sonst kehre zurück,ohne eine Routine
                             * in den VBL eingebunden zuhaben.
freier:
         lea     0(a0,d1),a1 * lade Adresse des freien Slots nach
                             * A1
         move.l #neuervbl,(a1) * installiere Routine im VBL
         rts                 * und kehre zurück

neuervbl:  .
           .
           .
        rts
```

Wenn Sie die Routine nicht mehr benötigen, reicht es, in den Slot eine Null zu laden (freie Slots sind durch eine Null gekennzeichnet).

Falls die acht freien Slots nicht ausreichen, verlegen Sie einfach die Slots mit den Routinen auf eine andere Adresse und stellen Sie die »Systemvariablen *nvbls* ($454) und *_vblqueue* ($456) dementsprechend um.

Vektoren

siehe »Exceptions

Vektoren

siehe »Exceptions

VDI

Engl., Abk. für "Virtual Device Interface", zu deutsch virtueller Gerätetreiber. VDI übernimmt auf dem Atari alle Grafik Ein- und Ausgabe für »GEM. VDI kann eine ganze Reihe von Geräten ansteuern, Bildschirme, Drucker Plotter, Kamerafilme (!), Mäuse, Grafiktabletts usw. usf. Für die aufgezählten Geräte werden le-

diglich entsprechende Gerätetreiber benötigt. Die Ansteuerung ist vom verwendeten Ausgabegerät unabhängig, sofern »Normalized Device Coordinates (Abk. NDC) verwandt wurden (nicht jedes Gerät beherrscht jede Ausgabeform, so macht ein Form Feed auf einem »Bildschirm ja auch wenig Sinn). Die Umrechnung von NDC in die gerätespezifischen »Raster Koordinaten (sofern letztere nicht gleich verwendet werden) wird von »GDOS (Graphics Device Operating System) übernommen.

VDI bietet eine Fülle von Routinen zur Grafikausgabe für (fast) jeden erdenklichen Zweck. Was auf anderen Rechnern erst vom Programmierer implementiert werden muß, VDI hat es bereits! Es existieren 118 Routinen, die in sieben Bereiche eingeteilt werden.

Die Bereiche sind:
- Routinen zum Manipulieren der »Workstation
- Routinen zur Text- und Grafikausgabe
- Routinen zum Verändern von Text- und Grafikattributen
- Raster-Routinen
- Eingabe-Routinen
- Abfrage-Routinen
- Escape-Sequenzen

Im folgenden erhalten Sie einen Überblick über alle Routinen, geordnet nach dem Verwendungsbereich. Zu jeder Routine existiert eine Beschreibung als eigenes Stichwort.

Workstation-Routinen

- »*v_clrwk()*	»Workstation löschen
- »*v_clsvwk()*	»virtual Workstation schließen
- »*v_clswk()*	»Workstation schließen
- »*v_opnvwk()*	»virtual Workstation öffnen
- »*v_opnwk()*	»Workstation öffnen
- »*v_updwk()*	»Workstation auf neuesten Stand bringen (update)

Ausgabe-Routinen

- »*v_arc()*	Kreisbogen (ausschnittweise) zeichnen
- »*v_bar()*	Gefüllten Balken zeichnen
- »*v_circle()*	Kreis zeichnen
- »*v_cellarray()*	Gitter zeichnen
- »*v_contourfill()*	beliebige umrandete Fläche mit Muster füllen
- »*v_ellarc()*	elliptischen Kreisbogenausschnitt zeichnen
- »*v_ellipse()*	Ellipse zeichnen
- »*v_ellpie()*	Ellipsenausschnitt zeichnen
- »*v_fillarea()*	Gefülltes Gebiet zeichnen

- »*v_gtext()* Grafiktext ausgeben (beliebige X- Y-Koordinate)
- »*v_justified()* Grafiktext mit Randausgleich ausgeben
- »*v_pieslice()* Kreis zeichnen
- »*v_pline()* mehrere zusammenhängende Linien zeichnen
- »*v_pmarker()* Markierung zeichnen
- »*v_rbox()* Rechteck mit abgerundeten Ecken zeichnen
- »*v_recfl()* Rechteck mit Muster ausfüllen
- »*v_rfbox()* Rechteck mit abgerundeten Ecken mit Muster füllen
- »*vs_clip()* Grafikausgabe auf rechteckigen Ausschnitt begrenzen (»Clipping)

Attribute-Routinen

- »*vs_color()* Farbpalette setzen
- »*vsf_color()* Füllfarbe definieren
- »*vsf_interior()* Stil des Füllmusters setzen
- »*vsf_perimeter()* Sichtbarkeit der Musterumrahmung setzen
- »*vsf_style()* Füllmuster festlegen
- »*vsf_updat()* Benutzer-definiertes Füllmuster setzen
- »*vsl_color()* Farbe der Linien setzen
- »*vsl_ends()* Art des Endes der Linien setzen
- »*vsl_type()* Linienart setzen
- »*vsl_udsty()* Benutzer-definiertes Linienmuster setzen
- »*vsl_width()* Breite der Linien setzen
- »*vsm_color()* Farbe der Marker setzen
- »*vsm_height()* Größe der Marker setzen
- »*vsm_type()* Markertyp setzen
- »*vst_alignment()* Bündigkeit des Grafiktextes setzen
- »*vst_color()* Farbe des Grafiktextes setzen
- »*vst_effects()* Texteffekte setzen
- »*vst_font()* Zeichensatz wählen
- »*vst_height()* Zeichengröße absolut setzen
- »*vst_point()* Zeichengröße in Punkten setzen
- »*vst_rotation()* Richtung der Grafiktextausgabe setzen
- »*vswr_mode()* Schreibmodus setzen

Raster-Routinen

- »*v_get_pixel()* Farbe eines »Pixels abfragen
- »*vr_trnfm()* »Form transformieren
- »*vro_cpyfm()* Raster-Kopier-Routine
- »*vrt_cpyfm()* Raster-Kopier-Routine von Monochrom nach Farbe

Eingabe-Routinen

- »*v_hide_c()*	Mauszeiger unsichtbar machen
- »*v_show_c()*	Mauszeiger sichtbar machen
- »*vex_butv()*	Mausknopf-Vektor setzen
- »*vex_curv()*	Cursor-Vektor setzen
- »*vex_motv()*	Maus-Vektor setzen
- »*vq_key_s()*	Fragt Zustand der Tastatur ab
- »*vq_mouse()*	Zustand des Mausknopfs abfragen
- »*vrq_choice()*	Wartet, bis eine Funktionstaste gedrückt wurde
- »*vrq_locator()*	Fragt Position des Mauscursors ab und zeigt ihn an
- »*vrq_string()*	Liest String ein
- »*vrq_valuator()*	Liest den Wert des Valuators
- »*vsc_form()*	Setzt eine eigene Form für den »Mauszeiger
- »*vsin_mode()*	Setzt den Eingabemodus
- »*vsm_choice()*	Fragt Status der Funktionstasten ab
- »*vsm_locator()*	Fragt Position des Mauscursors ab
- »*vsm_string()*	Liest String ein, bis keine Daten mehr verfügbar sind
- »*vsm_valuator()*	Fragt den Valuator ab

Abfrage-Routinen

- »*vq_cellarray()*	Fragt mit »*v_cellarray()* gesetzte Werte ab
- »*vq_color()*	Fragt gesetzte Farbe ab
- »*vq_extnd()*	Erweiterte Abfrage für alles mögliche
- »*vqf_attributes()*	Attribute für Füll-Parameter abfragen
- »*vqin_mode()*	Eingabe-Modus abfragen
- »*vql_attributes()*	Linien-Attribute abfragen
- »*vqm_attributes()*	Marker-Attribute abfragen
- »*vqt_attributes()*	Grafik-Text-Attribute abfragen
- »*vqt_extent()*	Erweiterte Abfrage für Text-Attribute
- »*vqt_fontinfo()*	Informationen über »Font abfragen
- »*vqt_name()*	Fontname abfragen
- »*vqt_width()*	Zeichenbreite abfragen

Escape-Sequenzen

- »*v_curdown()*	Bewegt den Textcursor eine Zeile nach unten
- »*v_curhome()*	Bewegt den Textcursor in die linke obere Ecke
- »*v_curleft()*	Bewegt den Textcursor eine Spalte nach links
- »*v_curright()*	Bewegt den Textcursor eine Spalte nach rechts
- »*v_curtext()*	Gibt Grafiktext an Cursorposition aus

- »*v_curup()* Bewegt den Textcursor eine Zeile nach oben
- »*v_eeol()* Löscht Zeile bis Zeilenende vom Cursor ab
- »*v_eeos()* Löscht ab Cursor bis »Bildschirmende
- »*v_enter_cur()* Umschalten auf Textmodus
- »*v_exit_cur()* Umschalten von Textmodus auf Grafikmodus
- »*v_rmoff()* Invertierte Textausgabe abschalten
- »*v_rvon()* Invertierte Textausgabe einschalten
- »*vq_chcells()* Anzahl der möglichen Buchstaben auf dem »Bildschirm abfragen
- »*vs_curadress()* Cursor direkt positionieren

Wie bei »AES auch, gibt es für VDI einen Parameterblock. Dieser hat folgenden Aufbau:

Offs.	Typ	Bedeutung der Werte im VDI-Parameterblock
0	long	Adresse des **contrl**-Arrays
4	long	Adresse des **intin**-Arrays
8	long	Adresse des **ptsin**-Arrays
12	long	Adresse des **intout**-Arrays
16	long	Adresse des **ptsout**-Arrays

Das **contrl**-Array besteht aus Worten. Sein Aufbau:

Offs.	Typ	Bedeutung der Werte im **contrl**-Array
0	word	Opcode-Nummer für die angesprochene VDI-Routine
2	word	Anzahl der Werte (Worte) im **ptsin**-Array. Da dies aus Wertepaaren besteht (Points in) ist dies die Anzahl der X-Y-Koordinatenpaare mal zwei
4	word	Die Länge des **ptsout**-Arrays (Ausgabeparameter!)
6	word	Die Länge des **intin**-Array
8	word	Die Länge des **intout**-Arrays (Ausgabeparameter!)
10	word	Identifikationsnummer einer Unterfunktion (dabei handelt es sich entweder um eine »Generalized Drawing-Primitive oder eine Escape-Routine)
12	word	Die Nummer des Ausgabegerätes (wird von der Routine »*v_opnvwk()* zurückgegeben und im weiteren als »handle der »virtual Workstation bezeichnet).
14-n	word	Abhängig vom Opcode

Die Arrays '**ptsin**' und '**intin**' werden benötigt, um dort die Eingabeparameter abzulegen. Welche Bedeutung die Eingabeparameter haben, ist für jede VDI-Routine gesondert erklärt. Die Arrays '**ptsout**' und '**intout**' sind für die Ausgabeparameter vorgesehen. Dort legt VDI **alle** Rückgabewerte ab. Die Länge der Arrays (welche aus Worten bestehen) ist für jede Routine unterschiedlich. Es muß jedoch soviel Platz vorhanden sein, um die Routine, die die meisten Werte benötigt, aufrufen zu können. Der Platzbedarf geht aus der Beschreibung der einzelnen VDI-Routinen hervor.

Bevor irgendeine der VDI-Routinen benutzt werden kann, muß erst eine »virtual Workstation ("virtuelle Arbeitsstation") geöffnet werden, um die Gerätenummer zu bekommen, die **alle** VDI-Routinen (außer »*v_opnwk()* und »*v_opnvwk()*) benötigen. Dies geschieht mit der Routine »*v_opnvwk()*.

Auf der Ebene der Maschinensprache sieht ein Aufruf einer VDI-Routine so aus (nachdem die Parameter für das **contrl**- **intin**- und **ptsin**-Array gesetzt wurden):

```
CALL_VDI
        MOVE.L    #vdi_paramblk,D1    ; Die Adresse des VDI-Pa-
                                      ; rameterblocks muß in
                                      ; das D1-Register
        MOVE.W    #$73,D0             ; Kennummer der VDI-Rou-
                                      ; tinen
        TRAP      #2                  ; Aufruf der Routine
        ...                           ; und weiter im Text...
```

Die Ausgabewerte kann man nun im **intout**- und/oder **ptsout**-Array abholen. Die Anzahl der Rückgabewerte steht in **contrl(2)** (für das **ptsout**-Array) und in **contrl(4)** (für das **intout**-Array).

VT52

Abk.: Virtual Tabulation 5.2. Unter VT52 versteht man ein Terminal, das bestimmte »Escape-Sequenzen auf eine genau festgelegte Weise verarbeitet. Das *eingebaute* Terminal des Atari ST versteht diese Steuerzeichen und noch einige Erweiterungen (»Bildschirmsteuerung).

V_arc

Zum »VDI gehörende Routine für Grafikausgabe

```
$0B = 11        v_arc(handle, x, y, radius, begang, endang)
word                  handle, x, y, radius, begang, endang;
```

Diese Funktion zeichnet einen Kreisausschnitt um den Punkt 'x', 'y' mit dem »Radius 'radius' vom Anfangswinkel 'begang' bis 'endang'. Der »Radius wird in »Pixeln angegeben. Nicht alle Ausgabegeräte beherrschen diese Funktion.

Die Ausgabe wird von folgenden Routinen beeinflußt:
- »*vsl_color()*
- »*vsl_ends()*
- »*vsl_type()*
- »*vsl_width()*
- »*vswr_mode()*

Eingabeparameter:

word handle	Handle der »virtual Workstation für Ausgabe
word x	X-Koordinate des Mittelpunkts des Kreisausschnitts
word y	Y-Koordinate des Mittelpunkts des Kreisausschnitts
word radius	Radius des Kreisausschnitts
word begang	Anfangswinkel des Kreisausschnitts
word endang	Endwinkel des Kreisausschnitts

```
contrl(0)   =  11
contrl(1)   =   4
contrl(3)   =   2
contrl(5)   =   2
contrl(6)   =  handle

intin(0)    =  begang
intin(1)    =  endang

ptsin(0)    =  x
ptsin(1)    =  y
ptsin(6)    =  radius
```

Ausgabeparameter: keine

```
contrl(2)   =   0
contrl(4)   =   0
```

V_bar

Zum »VDI gehörende Routine für Grafikausgabe

```
$0B = 11        v_bar(handle, pxyarray)
word                  handle;
word *                         pxyarray;
```

Diese Routine zeichnet einen Balken. Ein Balken ist ein ausgefülltes Rechteck. Der Inhalt des Balkens wird durch das festgelegte Füllmuster beeinflußt. Der Rand wird entsprechend dem festgelegten Linientyp und der festgelegten Liniendicke gezeichnet.

wenn mittels »*vsf_perimeter()* der Rand auf "sichtbar" eingestellt wurde.

Die Ausgabe wird von folgenden Routinen beeinflußt:
- »*vsf_color()*
- »*vsf_interior()*
- »*vsf_perimeter()*
- »*vsf_style()*
- »*vsl_type()*
- »*vsl_width()*
- »*vswr_mode()*

Eingabeparameter:

word handle	Handle der »virtual Workstation für Ausgabe
word *pxyarray	Zeiger auf ein vier Worte umfassendes Array welches die Koordinaten von zwei X-Y-Punkten enthält. Das erste Paar gibt ein beliebigen Eckpunkt des zu zeichnenden Balkens an, das zweite Paar den diagonal gegenüberliegenden Eckpunkt.

```
contrl(0)   =  11
contrl(1)   =   2
contrl(3)   =   0
contrl(5)   =   1
contrl(6)   =   handle

ptsin(0)    =  pxyarray[0]
ptsin(1)    =  pxyarray[1]
ptsin(2)    =  pxyarray[2]
ptsin(3)    =  pxyarray[3]
```

Ausgabeparameter: keine

```
contrl(2)   =   0
contrl(4)   =   0
```

V_bit_image

Zum »VDI gehörende Routine für Drucker

```
$05 = 5        v_bit_image(handle, filename, aspect, x_scale,
word                       handle,           aspect, x_scale;
char *                            filename;
                           y_scale, h_align, v_align, xyarray)
word                       y_scale, h_align, v_align;
word *                                                xyarray;
```

Mit dieser Funktion kann ein »Bit-Image auf einem Drucker ausgegeben werden. Dazu muß eine Datei vorhanden sein, die die entsprechenden Informationen enthält: eine solch Datei wird von verschiedenen Malprogrammen (z. B. GEM Paint) erzeugt (zum Dateiformat siehe »Bit Image).

Für die Ausgabe kann gewählt werden, wie das Bild auf der Drukkerseite ausgerichtet werden soll, in welchem Maßstab es ausgedruckt wird und ob es in X- oder Y-Richtung verzerrt dargestellt wird.

Eingabeparameter:

word handle	Handle der »virtual Workstation für Ausgabe
char *filename	Zeiger auf einen String mit dem Dateinamen
word aspect	Dieses Flag zeigt an, wie das X- Y-Verhältnis des »Bit-Image behandelt werden soll: 0 = Keinerlei Transformation vornehmen, ein »Pixel auf dem Quellgerät bleibt genau ein »Pixel auf dem Zielgerät. 1 = Verhältnis der Pixelgrößen berücksichtigen (ein Kreis bleibt ein Kreis, ein Quadrat bleibt ein Quadrat) 2 = Bild so anpassen, daß ein volles Bit-Image-Bild auf eine volle Druckseite abgebildet wird (Anm.: die Werte null und zwei können zu einer Verzerrung des Bildes führen)
word x_scale	0 = Maßstabsveränderung wird mit Bruchzahlen durchgeführt (in x-Richtung) 1 = Maßstabsveränderung wird mit Ganzzahlen durchgeführt (in x-Richtung)
word y_scale	0 = Maßstabsveränderung wird mit Bruchzahlen durchgeführt (in y-Richtung) 1 = Maßstabsveränderung wird mit Ganzzahlen durchgeführt (in y-Richtung)
word h_align	Horizontale Ausrichtung: 0 = Bild wird links ausgerichtet 1 = Bild wird zentriert 2 = Bild wird rechts ausgerichtet

```
word v_align        Vertikale Ausrichtung:
                    0 = Ausrichtung erfolgt nach oben
                    1 = Bild wird mittig ausgerichtet
                    2 = Bild wird nach unten ausgerichtet

word *xyarray       Ein aus vier Worten bestehendes Array:
                    xyarray(0) = X-Koordinate der linken oberen
                                 Ecke
                    xyarray(1) = Y-Koordinate der linken oberen
                                 Ecke
                    xyarray(2) = X-Koordinate der rechten unteren
                                 Ecke
                    xyarray(3) = Y-Koordinate der rechten unteren
                                 Ecke

contrl(0)   =   5
contrl(1)   =   0,1 oder 2:  0 = Linken oberen Eckpunkt und rech-
                             ten unteren Eckpunkt aus dem Header
                             der »Bit-Image Datei entnehmen
                             1 = Oberen linken Eckpunkt aus dem
                             ptsin-Array entnehmen, unteren rech-
                             ten aus der »Bit-Image Datei
                             2 = Oberen linken und unteren rech-
                             ten Eckpunkt aus dem ptsin-Array ho-
                             len
contrl(3)   =  Länge des Dateinamens plus Fünf
contrl(5)   =  23
contrl(6)   =  handle

intin(0)    =  aspect
intin(1)    =  x_scale
intin(2)    =  y_scale
intin(3)    =  h_align
intin(4)    =  v_align
intin(5)    =  Erstes Zeichen des Dateinamens
.
.
.
intin(n)    =  N. Zeichen des Dateinamens

ptsin(0)    =  xyarray[0]
ptsin(1)    =  xyarray[1]
ptsin(2)    =  xyarray[2]
ptsin(3)    =  xyarray[3]
```

Ausgabeparameter:

```
contrl(2)   =  0
contrl(4)   =  0
```

V_cellarray

Zum »VDI gehörende Routine für Grafikausgabe

```
$0A = 10        v_cellarray(handle, pxyarray, row_length, el_used,
word                        handle,           row_length, el_used;
word *                            pxyarray;
                            num_rows, wrt_mode, colarray)
word                        num_rows, wrt_mode, colarray;
```

Mit dieser Routine ist es möglich, auf dem Ausgabegerät ein Gitter zu zeichnen. Im Array 'pxyarray' stehen dabei die Koordinaten der beiden Ecken, und das Gitter besteht aus der angegebenen Anzahl von Spalten und Zeilen. Diese bilden ein Anzahl von Rechtekken. Für jedes dieser Rechtecke kann eine andere Farbe angegeben werden.

Leider ist diese Routine auf dem Atari nicht verfügbar (siehe »*v_opnvwk()*). Statt eines Gitters wird nur ein Rechteck in der aktuellen Linienfarbe und Linienbreite gezeichnet, so daß sich ein Beschreibung erübrigt.

V_circle

Zum »VDI gehörende Routine für Grafikausgabe

```
$0B = 11        v_circle(handle, x, y, radius)
word                     handle, x, y, radius;
```

Diese Routine zeichnet ein (mit einem Muster gefüllten) Kreis um den Punkt 'x', 'y' mit dem »Radius 'radius'. Der »Radius wird in »Pixeln angegeben. Nicht alle Ausgabegeräte beherrschen diese Funktion.

Die Ausgabe wird von folgenden Routinen beeinflußt:
- »*vsf_color()*
- »*vsf_interior()*
- »*vsf_perimeter()*
- »*vsf_style()*
- »*vswr_mode()*

Eingabeparameter:

word handle	Handle der »virtual Workstation für Ausgabe
word x	X-Koordinate des Kreismittelpunktes
word y	Y-Koordinate des Kreismittelpunktes
word radius	Radius des Kreises

```
contrl(0)   =  11
contrl(1)   =   3
contrl(3)   =   0
contrl(5)   =   4
contrl(6)   =  handle

ptsin(0)    =  x
ptsin(1)    =  y
ptsin(6)    =  radius
```

Ausgabeparameter: keine

```
contrl(2)   =   0
contrl(4)   =   0
```

V_clear_disp_list

Zum »VDI gehörende Routine für Drucker

```
$05 = 5          v_clear_disp_list(handle)
word                                handle;
```

Hiermit können Sie den Puffer, in dem die noch auszuführenden Kommandos für den Drucker stehen, löschen. Diese Routine macht dasselbe wie »*v_clrwk()*, führt aber keinen Seitenvorschub aus.

Eingabeparameter:

```
word handle        Handle der »virtual Workstation für Ausgabe

contrl(0)   =   5
contrl(1)   =   2
contrl(3)   =   0
contrl(5)   =  22
contrl(6)   =  handle
```

Ausgabeparameter:

```
contrl(2)   =  0
contrl(4)   =  0
```

V_clrwk

Zum »VDI gehörende Routine für Grafikausgabe

```
$03 = 3          v_clrwk(handle)
word                       handle;
```

Mit dieser Routine wird die Arbeitsstation gelöscht (auf dem

Atari ist das identisch mit dem Löschen des Bildschirm). Handelt es sich bei der »Workstation um einen Drucker, so wird ein Seitenvorschub ausgeführt. Bei einem Metafile wird die Funktionsnummer ausgegeben.

Eingabeparameter:

word handle Handle der Workstation für Ausgabe

```
contrl(0)   =  3
contrl(1)   =  0
contrl(3)   =  0
contrl(6)   =  handle
```

Ausgabeparameter:

```
contrl(2)   =  0
contrl(4)   =  0
```

V_clsvwk

Zum »VDI gehörende Routine für Grafikausgabe

$65 = 101 v_clsvwk(handle)
word handle;

Diese Routine schließt die »virtuelle Arbeitsstation wieder. Damit kann keine weitere Ausgabe auf diese Workstation mehr gehen. Die Nummer (handle) wird wieder freigegeben. Es ist sinnvoll, vor Beendigen eines Programms **alle** geöffneten Arbeitsstationen wieder zu schließen, da Nachfolgeprogramme davon ja auch Gebrauch machen wollen.

Eingabeparameter:

word handle Handle der »virtual Workstation für Ausgabe

```
contrl(0)   =  101
contrl(1)   =    0
contrl(3)   =    0
contrl(6)   =  handle
```

Ausgabeparameter:

```
contrl(2)   =  0
contrl(4)   =  0
```

V_clswk

Zum »VDI gehörende Routine für Grafikausgabe

```
$02 = 2          v_clswk(handle)
word                      handle;
```

Diese Routine ist nur der Vollständigkeit halber erwähnt. Denn das, was bei »*v_opnwk()* steht, trifft auch hier zu: da keine »Workstation geöffnet werden kann, ist es auch wenig sinnvoll, diese wieder zu schließen!

Eingabeparameter:

```
word handle        Handle der »virtual Workstation für Ausgabe

contrl(0)   =   2
contrl(1)   =   0
contrl(3)   =   0
contrl(6)   =  handle
```

Ausgabeparameter:

```
contrl(2)   =  0
contrl(4)   =  0
```

V_contourfill

Zum »VDI gehörende Routine für Grafikausgabe

```
$67 = 103          v_contourfill(handle, x, y, index)
word                             handle, x, y, index;
```

Diese Routine dient dem Ausfüllen beliebig geformter Flächen. Das Ausfüllen geschieht entweder, bis der Bildschirmrand erreicht ist **oder** in jeder Zeile bis eine andere Farbe erreicht wird. Die Routine startet mit dem Füllen an den Koordinaten 'x', 'y' und fährt solange fort, bis eine Farbe 'index' erreicht ist. Wenn 'index' negativ ist, so wird als begrenzende Farbe jede Farbe genommen, die von der an der Stelle 'x', 'y' verschieden ist.

Die Ausgabe wird von folgenden Routinen beeinflußt:
- »*vsf_color()*
- »*vsf_interior()*
- »*vsf_perimeter()*
- »*vsf_style()*
- »*vswr_mode()*

Nicht jedes Gerät beherrscht diese Funktion.

Eingabeparameter:

word handle	Handle der »virtual Workstation für Ausgabe
word x	X-Position, ab der das Gebiet gefüllt wird
word y	Y-Position, ab der das Gebiet gefüllt wird
word index	Farbe, die als Gebietsgrenze verwendet wird. Wenn 'index' = -1, so ist die begrenzende Farbe jede Farbe, die anders ist als die Farbe am Punkt 'x', 'y'.

```
contrl(0)   =  103
contrl(1)   =  1
contrl(3)   =  1
contrl(6)   =  handle

intin(0)    =  index

ptsin(0)    =  x
ptsin(1)    =  y
```

Ausgabeparameter:

```
contrl(2)   =  0
contrl(4)   =  0
```

V_curdown

Zum »VDI gehörende Escape-Routine für Text

```
$05 = 5          v_curdown(handle)
word                        handle;
```

Mit dieser Routine wird der Text-Cursor um eine Zeile nach unten bewegt (ohne die Spalten-Position zu verändern). Befindet sich der Cursor bereits in der untersten Zeile, so geschieht nichts.

Siehe auch »Bildschirmsteuerung.

Eingabeparameter:

word handle	Handle der »virtual Workstation für Ausgabe

```
contrl(0)   =  5
contrl(1)   =  0
contrl(3)   =  0
contrl(5)   =  5
contrl(6)   =  handle
```

Ausgabeparameter:

contrl(2) = 0
contrl(4) = 0

V_curhome

Zum »VDI gehörende Escape-Routine für Text

$05 = 5 **v_curhome(handle)**
word handle;

Mit dieser Routine wird der Text-Cursor in die obere linke Ecke des »Bildschirms verfrachtet.

Siehe auch »Bildschirmsteuerung.

Eingabeparameter:

word handle Handle der »virtual Workstation für Ausgabe

contrl(0) = 5
contrl(1) = 0
contrl(3) = 0
contrl(5) = 8
contrl(6) = handle

Ausgabeparameter:

contrl(2) = 0
contrl(4) = 0

V_curleft

Zum »VDI gehörende Escape-Routine für Text

$05 = 5 **v_curleft(handle)**
word handle;

Der Text-Cursor wird mit dieser Routine um eine Spalte nach links bewegt (Zeilenposition bleibt unverändert). Befindet sich der Cursor am linken Rand, so geschieht nichts.

Siehe auch »Bildschirmsteuerung.

Eingabeparameter:

word handle Handle der »virtual Workstation für Ausgabe

```
contrl(0)   =  5
contrl(1)   =  0
contrl(3)   =  0
contrl(5)   =  7
contrl(6)   =  handle
```

Ausgabeparameter:

```
contrl(2)   =  0
contrl(4)   =  0
```

V_curright

Zum »VDI gehörende Escape-Routine für Text

```
$05 = 5          v_curright(handle)
word                        handle;
```

Hiermit kann der Text-Cursor um eine Spalte nach rechts bewegt werden, sofern er sich nicht bereits am rechten Rand befindet. Die Zeilenposition bleibt unverändert.

Siehe auch »Bildschirmsteuerung.

Eingabeparameter:

```
word handle        Handle der »virtual Workstation für Ausgabe

contrl(0)   =  5
contrl(1)   =  0
contrl(3)   =  0
contrl(5)   =  6
contrl(6)   =  handle
```

Ausgabeparameter:

```
contrl(2)   =  0
contrl(4)   =  0
```

V_curtext

Zum »VDI gehörende Escape-Routine für Text

```
$05 = 5          v_curtext(handle, &string)
word                       handle;
char *                             string;
```

Mit dieser Routine kann eine Zeichenkette an der aktuellen Position des Text-Cursors ausgegeben werden.

Die Darstellungsweise des Textes wird von folgenden Routinen beeinflußt:
- »*v_rvoff()*
- »*v_rvon()*

Eingabeparameter:

```
word handle        Handle der »virtual Workstation für Ausgabe

char *string       Adresse des auszugebenden Strings

contrl(0)   =  5
contrl(1)   =  0
contrl(3)   =  Anzahl der Zeichen im String
contrl(5)   =  12
contrl(6)   =  handle

intin(0)    =  Erstes Zeichen in den unteren acht Bit des Wortes
intin(2)    =  Zweites Zeichen in den unteren acht Bit des Wortes
.
.
.
intin(n)    =  n. Zeichen im String
```

Ausgabeparameter:

```
contrl(2)   =  0
contrl(4)   =  0
```

V_curup

Zum »VDI gehörende Escape-Routine für Text

```
$05 = 5          v_curup(handle)
word                     handle;
```

Mit dieser Routine wird der Text-Cursor um eine Zeile nach oben bewegt. Ist er in der obersten Zeile, so passiert nichts.

Siehe auch »Bildschirmsteuerung.

Eingabeparameter:

```
word handle        Handle der »virtual Workstation für Ausgabe

contrl(0)   =  5
contrl(1)   =  0
contrl(3)   =  0
contrl(5)   =  4
contrl(6)   =  handle
```

Ausgabeparameter:

contrl(2) = 0
contrl(4) = 0

V_dspcur

Zum »VDI gehörende Escape-Routine für Grafik

```
$05 = 5          v_dspcur(handle, x, y)
word                      handle, x, y;
```

Mit dieser Routine kann man einen Grafik-Cursor an einer bestimmten Position ausgeben. Die Form des Cursors ist diesselbe wie bei »*vrq_locator()*. Es ist möglich, diese Routine vor einem Aufruf von »*vsm_locator()* zu benutzen, um den Cursor sichtbar zu machen. Der Cursor wird im sog. XOR-Modus gezeichnet (Exklusiv-Oder), so daß ein Aufruf von »*v_rmcur()* diesen wieder entfernen kann.

Eingabeparameter:

word handle	Handle der »virtual Workstation für Ausgabe
word x	X-Position des Cursors
word y	Y-Position des Cursors

contrl(0) = 5
contrl(1) = 1
contrl(3) = 0
contrl(5) = 18
contrl(6) = handle

ptsin(0) = x
ptsin(0) = y

Ausgabeparameter:

contrl(2) = 0
contrl(4) = 0

V_eeol

Zum »VDI gehörende Escape-Routine für Text

```
$05 = 5          v_eeol(handle)
word                    handle;
```

Diese Routine löscht den Text von der Position des Text-Cursors

bis zum Ende der Zeile. Die Cursor-Position bleibt unverändert.

Siehe auch »Bildschirmsteuerung.

Eingabeparameter:

word handle Handle der »virtual Workstation für Ausgabe

```
contrl(0)   =  5
contrl(1)   =  0
contrl(3)   =  0
contrl(5)   =  10
contrl(6)   =  handle
```

Ausgabeparameter:

```
contrl(2)   =  0
contrl(4)   =  0
```

v_eeos

Zum »VDI gehörende Escape-Routine für Text

```
$05 = 5          v_eeos(handle)
word                    handle;
```

Diese Routine löscht den Text von der Position des Text-Cursors bis zum Ende des »Bildschirms. Die Cursor-Position bleibt unverändert.

Siehe auch »Bildschirmsteuerung.

Eingabeparameter:

word handle Handle der »virtual Workstation für Ausgabe

```
contrl(0)   =  5
contrl(1)   =  0
contrl(3)   =  0
contrl(5)   =  9
contrl(6)   =  handle
```

Ausgabeparameter:

```
contrl(2)   =  0
contrl(4)   =  0
```

V_ellarc

Zum »VDI gehörende Routine für Grafikausgabe

```
$0B = 11        v_ellarc(handle, x, y, xradius, yradius,
word                     handle, x, y, xradius, yradius;
                         begang, endang)
word                     begang, endang;
```

Diese Routine zeichnet einen elliptischen Bogen um die Koordinaten 'x' und 'y' mit dem Radius 'xradius' in X-Richtung und dem Radius 'yradius' in Y-Richtung mit dem Anfangswinkel 'begang' und dem Endwinkel 'endang'. Nicht alle Ausgabegeräte beherrschen diese Funktion.

Die Ausgabe wird von folgenden Routinen beeinflußt:
- »*vsl_color()*
- »*vsl_ends()*
- »*vsl_type()*
- »*vsl_width()*
- »*vswr_mode()*

Eingabeparameter:

word handle	Handle der »virtual Workstation für Ausgabe
word x	X-Koordinate des Mittelpunkts des elliptischen Bogens
word y	Y-Koordinate des Mittelpunkts des elliptischen Bogens
word xradius	Radius des Bogens in X-Richtung
word yradius	Radius des Bogens in Y-Richtung
word begang	Anfangswinkel des Bogens
word endang	Endwinkel des Bogens

```
contrl(0)   =  11
contrl(1)   =   2
contrl(3)   =   2
contrl(5)   =   6
contrl(6)   =  handle

intin(0)    =  begang
intin(1)    =  endang

ptsin(0)    =  x
ptsin(1)    =  y
ptsin(2)    =  xradius
ptsin(3)    =  yradius
```

Ausgabeparameter: keine

```
contrl(2)   =   0
contrl(4)   =   0
```

V_ellipse

Zum »VDI gehörende Routine für Grafikausgabe

```
$0B = 11        v_ellipse(handle, x, y, xradius, yradius)
word                      handle, x, y, xradius, yradius;
```

Diese Routine zeichnet eine (mit einem Muster gefüllte) Ellipse um den Punkt 'x', 'y' mit dem »Radius 'xradius' in X-Richtung und dem »Radius 'yradius' in Y-Richtung. Der »Radius wird in »Pixeln angegeben. Nicht alle Ausgabegeräte beherrschen diese Funktion.

Die Ausgabe wird von folgenden Routinen beeinflußt:
- »*vsf_color()*
- »*vsf_interior()*
- »*vsf_perimeter()*
- »*vsf_style()*
- »*vswr_mode()*

Eingabeparameter:

```
word handle        Handle der »virtual Workstation für Ausgabe
word x             X-Koordinate des Mittelpunkts der Ellipse
word y             Y-Koordinate des Mittelpunkts der Ellipse
word xradius       Radius der Ellipse in X-Richtung
word yradius       Radius der Ellipse in Y-Richtung
```

```
contrl(0)   =  11
contrl(1)   =   2
contrl(3)   =   0
contrl(5)   =   5
contrl(6)   =  handle

ptsin(0)    =  x
ptsin(1)    =  y
ptsin(2)    =  xradius
ptsin(3)    =  yradius
```

Ausgabeparameter: keine

```
contrl(2)   =   0
contrl(4)   =   0
```

V_ellpie

Zum »VDI gehörende Routine für Grafikausgabe

```
$0B = 11        v_ellpie(handle, x, y, xradius, yradius,
word                     handle, x, y, xradius, yradius;
                         begang, endang)
word                     begang, endang;
```

Diese Routine zeichnet einen elliptisches Tortenstück um die Koordinaten 'x' und 'y' mit dem Radius 'xradius' in X-Richtung und dem Radius 'yradius' in Y-Richtung mit dem Anfangswinkel 'begang' und dem Endwinkel 'endang'. Nicht alle Ausgabegeräte beherrschen diese Funktion.

Die Ausgabe wird von folgenden Routinen beeinflußt:
- »*vsf_color()*
- »*vsf_interior()*
- »*vsf_perimeter()*
- »*vsf_style()*
- »*vswr_mode()*

Eingabeparameter:

word handle	Handle der »virtual Workstation für Ausgabe
word x	X-Koordinate des Mittelpunkts des elliptischen Tortenstücks
word y	Y-Koordinate des Mittelpunkts des elliptischen Tortenstücks
word xradius	Radius des Tortenstücks in X-Richtung
word yradius	Radius des Tortenstücks in Y-Richtung
word begang	Anfangswinkel des Tortenstücks
word endang	Endwinkel des Tortenstücks

```
contrl(0)   =  11
contrl(1)   =   2
contrl(3)   =   2
contrl(5)   =   7
contrl(6)   =  handle

intin(0)    =  begang
intin(1)    =  endang

ptsin(0)    =  x
ptsin(1)    =  y
ptsin(2)    =  xradius
ptsin(3)    =  yradius
```

Ausgabeparameter: keine

contrl(2) = 0
contrl(4) = 0

v_enter_cur

Zum »VDI gehörende Escape-Routine für Text

$05 = 5 **v_enter_cur(handle)**
word handle;

Mit dieser Routine schaltet man vom Grafik-Modus in den Text-Modus. Da dies beim Atari ST keinen Unterschied ausmacht, wird hiermit lediglich der Text-Cursor in die linke obere Ecke gebracht.

Eingabeparameter:

word handle Handle der »virtual Workstation für Ausgabe

contrl(0) = 5
contrl(1) = 0
contrl(3) = 0
contrl(5) = 3
contrl(6) = handle

Ausgabeparameter:

contrl(2) = 0
contrl(4) = 0

v_exit_cur

Zum »VDI gehörende Escape-Routine für Text

$05 = 5 **v_exit_cur(handle)**
word handle;

Mit dieser Routine schaltet man vom Text-Modus in den Grafik-Modus. Diese Routine bewirkt nur auf Geräten etwas, die unterschiedliche Modi für Text und Grafik haben (beim Atari gibt es diese Unterschiede nicht).

Eingabeparameter:

word handle Handle der »virtual Workstation für Ausgabe

```
contrl(0)   =  5
contrl(1)   =  0
contrl(3)   =  0
contrl(5)   =  2
contrl(6)   =  handle
```

Ausgabeparameter:

```
contrl(2)   =  0
contrl(4)   =  0
```

V_fillarea

Zum »VDI gehörende Routine für Grafikausgabe

```
$09 = 9          v_fillarea(handle, count, pxyarray)
word                        handle, count;
word *                                      pxyarray;
```

Mit dieser Routine kann ein komplexes Polygon, ausgefüllt mit dem aktuellen Füllmuster, gezeichnet werden. Die Definition des Polygons geschieht wie bei »*v_pline()*, nur trägt »VDI dafür Sorge, daß der letzte Punkt mit dem ersten verbunden wird (damit eine geschlossene Fläche entsteht). Das Polygon muß mindestens aus zwei Punkten bestehen.

Wenn mit der Routine »*vsf_perimeter()* die Umrandung auf "sichtbar" gestellt wurde, so wir der Rand des Polygons mit der aktuellen Füllfarbe gefärbt. Wenn das Polygon keine Fläche hat, so ist es nur bei sichtbarer Umrandung zu sehen.

Wieviele Punkte das Polygon maximal haben darf, ist mit »*vq_extend()* zu erfahren.

Die Ausgabe wird von folgenden Routinen beeinflußt:
- »*vsf_color()*
- »*vsf_interior()*
- »*vsf_perimeter()*
- »*vsf_style()*
- »*vswr_mode()*

Eingabeparameter:

word handle	Handle der »virtual Workstation für Ausgabe
word count	Anzahl der (Eck-) Punkte des Polygons

word *pxyarray Zeiger auf ein Array, mit dem die Punkte des Polygons definiert werden. Jeder Punkt im Array wird durch zwei Worte festgelegt. So steht die X-Koordinate des ersten Punktes in pxyarray(0), die Y-Koordinate in pxyarray(1) usw.

```
contrl(0)    =  9
contrl(1)    =  count
contrl(3)    =  0
contrl(6)    =  handle

ptsin(0)     =  X-Koordinate des ersten Punktes
ptsin(1)     =  Y-Koordinate des zweiten Punktes
.
.
.
ptsin(2n-2)  =  X-Koordinate des letzten Punktes (n = count)
ptsin(2n-1)  =  Y-Koordinate des letzten Punktes (n = count)
```

Ausgabeparameter:

```
contrl(2)    =  0
contrl(4)    =  0
```

V_form_adv

Zum »VDI gehörende Escape-Routine für den Drucker

```
$05 = 5          v_form_adv(handle)
word                        handle;
```

Mit dieser Routine kann man einen Seitenvorschub bei Druckern auslösen, falls es sich bei der »Workstation um einen Drucker handelt. Löst man den Seitenvorschub mit »*v_clrwk()* aus, so werden zusätzlich noch alle Kommandos im Druckerpuffer ausgeführt. Bei *v_form_adv()* unterbleibt das.

Eingabeparameter:

word handle Handle der »virtual Workstation für Ausgabe

```
contrl(0)    =   5
contrl(1)    =   0
contrl(3)    =   0
contrl(5)    =  20
contrl(6)    =  handle
```

Ausgabeparameter:

```
contrl(2)   =  0
contrl(4)   =  0
```

V_get_pixel

Zum »VDI gehörende Routine für Grafikausgabe

```
$69 = 105       v_get_pixel(handle, x, y, &pel, &index)
word                        handle, x, y;
word *                                   pel,  index;
```

Hiermit können Sie den Farbindex eines beliebigen »Pixels auf dem Bildschirm feststellen. Der Farbindex NULL bezeichnet die Hintergrundfarbe. Bei monochromem Bild gibt es nur noch den Farbindex Eins. Für denselben Zweck gibt es auch einen »Line-A-Befehl.

Eingabeparameter:

word handle	Handle der »virtual Workstation für Ausgabe
word x	X-Koordinate des Pixels, dessen Farbindex gesucht wird
word y	Y-Koordinate des Pixels, dessen Farbindex gesucht wird
word *pel word *index	Zeiger auf Rückgabewerte (s. u.)

```
contrl(0)   =  105
contrl(1)   =    1
contrl(3)   =    0
contrl(6)   =  handle

ptsin(0)    =  x
ptsin(1)    =  y
```

Ausgabeparameter:

word pel	Wert des Pixels
word index	Farbindex des Pixels

```
contrl(2)   =  0
contrl(4)   =  2
```

```
intout(0)   =  pel
intout(1)   =  index
```

V_gtext

Zum »VDI gehörende Routine für Grafikausgabe

```
$08 = 8          v_gtext(handle, x, y, string)
word                     handle, x, y;
char *                                 string;
```

Diese Routine dient dem Ausgeben einer Zeichenkette (Textstring). Anders als bei den entsprechenden Ausgaberoutinen des »BIOS und des »GEMDOS kann der Text an einer beliebigen Position (X-Y-Koordinate) ausgegeben werden. Außerdem ist es möglich, die Schriftgröße (»*vst_height()* oder »*vst_point()*) einzustellen oder die Schriftart (»*vst_effects()*) zu ändern. Ja, sogar die Richtung der Textausgabe kann beeinflußt werden (mit »*vst_rotation()*, aber nur in 90-Grad-Winkeln). Und: diese Routine berücksichtigt »Clipping, d. h. die Textausgabe läßt sich auf einen bestimmten Bereich (z. B. »Fenster) begrenzen. Was darüber hinausgeht, wird abgeschnitten. Die »BIOS-Routine schreibt gnadenlos auch über Fensterränder hinaus. Daher sollte für Textausgabe in Fenstern immer *v_gtext()* (oder »*v_justified()*) benutzt werden (außer, man weiß genau, was man tut).

Die Ausgabe wird von folgenden Routinen beeinflußt:
- »*vst_color()*
- »*vst_effects()*
- »*vst_font()*
- »*vst_height()*
- »*vst_rotation()*

Eingabeparameter:

word handle	Handle der »virtual Workstation für Ausgabe
word x	X-Koordinate, an der der String ausgegeben wird
word y	Y-Koordinate, an der der String ausgegeben wird
char *string	Adresse des auszugebenden Strings. Die 8-Bit-Werte der »ASCII-Zeichen des Strings werden in die 16-Bit-Werte des intin-arrays gepackt, die oberen 8 Bit sind Null. Die maximale Stringlänge ergibt sich aus der maximalen Größe des intin-Arrays (siehe »*vq_extend()*).

```
contrl(0)   =  8
contrl(1)   =  1
contrl(3)   =  Anzahl der Zeichen im String
contrl(6)   =  handle

intin(0)    =  - erstes Zeichen des Strings
intin(1)    =  - zweites Zeichen im String
...
intin(n)    =  - Letztes Zeichen im String

ptsin(0)    =  x
ptsin(1)    =  y
ptsin(2)    =  length
ptsin(3)    =  0
```

Ausgabeparameter: keine

```
contrl(2)   =  0
contrl(4)   =  0
```

V_hardcopy

Zum »VDI gehörende Routine für den Drucker

```
$05 = 5          v_hardcopy(handle)
word                        handle;
```

Hiermit können Sie, genau wie mit der »XBIOS-Routine *Scrdmp()*, den »Bildschirm als Hardcopy auf Ihrem Drucker ausdrucken, sofern Sie einen vollständig Epson-Fx-80-kompatiblen Drucker haben.

Die Routine springt über den Vektor, der bei $502 im Speicher steht. Für die verschiedensten Drucker gibt es bereits Treiber, die sich dies zunutze machen und den Vektor auf eine residente Routine richten, die eine angepaßte Hardcopy vornimmt.

Eingabeparameter:

```
word handle        Handle der »virtual Workstation für Ausgabe

contrl(0)   =   5
contrl(1)   =   0
contrl(3)   =   0
contrl(5)   =  17
contrl(6)   =  handle
```

Ausgabeparameter:

```
contrl(2)   =  0
contrl(4)   =  0
```

V_hide_c

Zum »VDI gehörende Routine für Grafikausgabe

```
$7B = 123       v_hide_c(handle)
word                     handle;
```

Mit dieser Routine kann der Mauscursor unsichtbar gemacht werden. Die Routine bewirkt exakt dasselbe wie folgender Aufruf:

```
        graf_mouse(256, 0);
word                256;
word *                   0;
```

Daher siehe auch »*graf_mouse()*!

Eingabeparameter:

```
word handle        Handle der »virtual Workstation für Ausgabe

contrl(0)   =  123
contrl(1)   =    0
contrl(3)   =    0
contrl(6)   =  handle
```

Ausgabeparameter:

```
contrl(2)   =  0
contrl(4)   =  0
```

V_justified

Zum »VDI gehörende Routine für Grafikausgabe

```
$0B = 11        v_justified(handle, x, y, string, length,
word                        handle, x, y,         length;
char *                                       string;
                            word_space, char_space)
word                        word_space, char_space;
```

Diese Routine gibt Text auf dem »Bildschirm aus, und zwar mit linkem und rechtem Randausgleich. Um die Zeichenkette zur benötigten Länge zu strecken werden die Buchstabenzwischenräume und/oder die Wortzwischenräume erweitert. Somit läßt sich leicht

Blocksatz mit Proportionalschrift auf dem »Bildschirm realisieren, sofern durch »*vst_load_fonts()* ein entsprechender »Font zur Verfügung steht.

Die Ausgabe wird von folgenden Routinen beeinflußt:
- »*vst_color()*
- »*vst_effects()*
- »*vst_font()*
- »*vst_height()*
- »*vst_rotation()*

Eingabeparameter:

word handle	Handle der »virtual Workstation für Ausgabe
word x	X-Koordinate, an der der String ausgegeben wird
word y	Y-Koordinate, an der der String ausgegeben wird
char *string	Adresse des auszugebenden Strings
word length	Länge die der String auf dem Bildschirm haben soll (in »Pixeln)
word word_space	'word_space' = 1: Längenausgleich geschieht durch Dehnen der Wortzwischenräume 'word_space' = 0: kein Längenausgleich durch dehnen der Wortzwischenräume
word char_space	'char_space' = 1: Längenausgleich geschieht durch Dehnen der Buchstabenzwischenräume 'char_space' = 0: kein Längenausgleich durch Dehnen der Buchstabenzwischenräume

```
contrl(0)   =  11
contrl(1)   =   2
contrl(3)   =   2 + Anzahl der Zeichen im String
contrl(5)   =  10
contrl(6)   =  handle

intin(0)    =  word_space
intin(1)    =  char_space
intin(2)    =  - erstes Zeichen des Strings
intin(3)    =  - zweites Zeichen im String
...
intin(n+1)  =  - Letztes Zeichen im String
```

```
ptsin(0)    =  x
ptsin(1)    =  y
ptsin(2)    =  length
ptsin(3)    =  0
```

Ausgabeparameter: keine

```
contrl(2)   =  0
contrl(4)   =  0
```

v_meta_extents

Zum »VDI gehörende Escape-Routine für Metafiles

```
$05 = 5          v_meta_extents(handle, min_x, min_y, max_x, max_y)
word                            handle, min_x, min_y, max_x, max_y;
```

Mit dieser Routine kann man in den Metafile-Header Informationen über die Größe der beschriebenen Grafik hineinschreiben (siehe »Metafile). Diese Information kann sehr nützlich sein, weil das Programm, welches das Metafile liest, Auskunft über den Umfang der Grafik bei der Darstellung bekommen kann, ohne alle Werte des Metafiles zu testen.

Eingabeparameter:

```
word handle        Handle der »virtual Workstation für Ausgabe

word min_x         Kleinsmöglicher X-Wert
word min_y         Kleinsmöglicher Y-Wert
word max_x         Größtmöglicher X-Wert
word max_y         Größtmöglicher Y-Wert
                   Diese Vier Werte beschreiben den Rahmen, in
                   dem die gesamte Grafik Platz findet.

contrl(0)   =   5
contrl(1)   =   2
contrl(3)   =   0
contrl(5)   =  98
contrl(6)   =  handle

intin(0)    =  min_x
intin(1)    =  min_y
intin(2)    =  max_x
intin(3)    =  max_y
```

Ausgabeparameter: keine

contrl(2) = 0
contrl(4) = 0

V_opnvwk

Zum »VDI gehörende Routine für Grafikausgabe

```
$64 = 100        v_opnvwk(work_in, &handle, work_out)
word *                    work_in,  handle, work_out;
```

Diese Routine ist die Voraussetzung für den Aufruf **aller** weiteren »VDI-Routinen. Sie gibt einen Wert zurück - den "handle" - der bei allen anderen Routinen stets den ersten Parameter abgibt. Dieser "handle" (zu deutsch "Griff") dient dazu, die jeweilige Arbeitsstation (engl.: Workstation) zu identifizieren. Daraus geht auch schon hervor, daß es möglich ist, mehrere dieser virtuellen Arbeitsstationen zu öffnen.

Dies ist deshalb interessant, weil jede dieser Workstation eigene Parameter für Grafikausgabe haben kann, unterschiedliche Bereiche für »Clipping beispielsweise, oder verschiedene Werte für Textausgabe, verschiedene Koordinatensysteme etc. So kann man für jedes Fenster eine eigene Arbeitsstation öffnen, damit man beim Umschalten auf ein anderes Fenster als Ausgabemedium nicht alle Attribute neu setzen muß. Allerdings hat jede Workstation Zugriff auf den **gesamten** Bildschirm.

Die Ein- und Ausgabeparameter sind dieselben wie bei »*v_opnwk()*. Allerdings benötigt *v_opnvwk()* noch die Identifikationsnummer des Ausgabegeräts. Dies ist beim Atari eine eins für den Bildschirm.

In dem Array 'work_in' (besteht aus elf Worten) müssen Voreinstellungen für verschiedene Attribute angegeben werden. Sinnvoll als erste Vorbesetzung ist eine eins, außer vielleicht bei 'work_in(10)'. Die Voreinstellungen sind wie folgt:

- Zeichenhöhe	Normale Zeichenhöhe
- Textwinkel	Null Grad
- Textausrichtung	Linke Basislinie
- Texteffekte	Keine
- Linienbreite	Normale Breite
- Markerhöhe	Normale Markerhöhe
- Linienenden	Rechteckig
- Schreibmodus	Überschreiben
- Eingabemodus	»Request
- Füllrand	Sichtbar
- Benutzer-definierte Linie	Durchgezogen

- Benutzer-definiertes Muster	Atari-Symbol
- Cursor	Abgeschaltet
- »Clipping	Abgeschaltet

ACHTUNG: Normalisierte Koordinaten (»Normalized Device Coordinates) können nur dann verwendet werden, wenn zu Beginn das Programm "GDOS.PRG" gebootet wurde! Siehe auch »GDOS

Eingabeparameter:

word handle — Handle einer bereits zuvor geöffneten »virtual Workstation. Diesen Wert bekommt man entweder mit »*v_opnwk()* heraus, oder mit »*graf_handle()*. Die Routine »*v_opnwk()* funktioniert auf dem Atari nicht, und so bleibt die zweite Methode. Da auf dem Atari der von »*graf_handle()* gelieferte Wert immer eins ist, kann man darauf auch verzichten...
Übrigens ist dieser Wert **auch** ein Rückgabeparameter!

word work_in() — Zeiger auf ein elf Worte umfassendes Array, in dem Werte für die Voreinstellung von Attributen stehen. Im folgenden die Bedeutung dieser elf Werte:

work_in(0) = Identifikationsnummer des Ausgabegeräts (1 = Bildschirm)
work_in(1) = Linientyp (siehe »*vsl_type()*)
work_in(2) = Linienfarbe (siehe »*vsl_color()*)
work_in(3) = Markertyp (siehe »*vsm_type()*)
work_in(4) = Markerfarbe (siehe »*vsm_color()*)
work_in(5) = Zeichensatz (beim Atari geht momentan nur eine eins, außer, das neue »GDOS wurde gebootet)
work_in(6) = Textfarbe (siehe »*vst_color()*)
work_in(7) = Füllstil (siehe »*vsf_interior()*)
work_in(8) = Füllmuster (siehe »*vsf_style()*)
work_in(9) = Füllfarbe (siehe »*vsf_color()*)
work_in(10) = Dieses Flag gibt an, ob »normalisierte oder »Raster-Koordinaten verwendet werden sollen:
0 = Normalisierte Koordinaten
1 = Reserviert
2 = »Rasterkoordinaten

ACHTUNG: Normalisierte Koordinaten (»NDC) können nur dann verwendet werden, wenn »GDOS beim Start des Betriebssystems installiert worden ist! Siehe auch »GDOS!

Dann allerdings ist es auch möglich, die beiden Koordinatensystem gemischt zu verwenden, d. h. eine Workstation benutzt »NDC-Koordinaten, eine andere »Rasterkoordinaten.

word *handle
word *work_out Zeiger auf Variablen für Rückgabewerte (s. u.)

```
contrl(0)   =  100
contrl(1)   =    0
contrl(3)   =   11
contrl(6)   =  handle eines zuvor geöffneten Bildschirms (=eins)

intin(0)    =  work_in(0)
intin(1)    =  work_in(1)
.
.
.
intin(10)   =  work_in(10)
```

Ausgabeparameter:

word handle Die Nummer, mit der alle weiteren Ausgaben auf die soeben geöffnete virtuelle Arbeitsstation gelenkt werden können (falls ein Fehler vorgekommen ist, bekommt man eine NULL zurück) Dieser Wert wird für alle weiteren »VDI-Routinen benötigt!

word work_out(57) Ein 57 Worte umfassendes Array, in dem umfangreiche Angaben für die geöffnete virtuelle Arbeitsstation stehen. Die Bedeutung der Werte sind im Einzelnen (wobei die Parameter, die der Atari zurückgibt, in Klammern dahinter stehen):

work_out(0) Anzahl der adressierbaren »Pixel in X-Richtung. Ein Wert von 639 bedeutet, daß das Ausgabegerät »Pixel von 0 bis 639 setzen kann (beim Atari monochrom beispielsweise), also sind 640 »Pixel insgesamt vorhanden.

work_out(1) Anzahl der adressierbaren »Pixel in Y-Richtung.

work_out(2)	Ist das Ausgabegerät in der Lage, Bilder in einer präzisen Skala auszugeben? 0 = ja (beim Atari) 1 = nein
work_out(3)	Breite eines »Pixels in Micron (ein Micron = 1/1000 mm, 372 beim Atari)
work_out(4)	Höhe eines »Pixels in Micron (ein Micron = 1/1000 mm, 372 beim Atari)
work_out(5)	Anzahl der Schriftgößen (drei beim Atari), NULL bei kontinuierlicher Skala
work_out(6)	Anzahl der Linientypen (sieben beim Atari)
work_out(7)	Anzahl der Linienbreiten, NULL bei kontinuierlicher Skala (wie etwa beim Atari)
work_out(8)	Anzahl der Markertypen (sechs beim Atari)
work_out(9)	Anzahl der Markergrößen, NULL bei kontinuierlicher Skala (acht beim Atari)
work_out(10)	Anzahl der unterstützten Zeichensätze (leider nur einen beim Atari, aber siehe auch »GDOS)
work_out(11)	Anzahl der Musterarten (24 beim Atari)
work_out(12)	Anzahl der Schraffurarten (12 beim Atari)
work_out(13)	Anzahl der vordefinierten Farben (beim Atari: zwei für monochrom)
work_out(14)	Anzahl der verfügbaren »Generalized Drawing Primitives (»GDP, zehn beim Atari)
work_out(15-24)	Die nächsten zehn Worte stellen eine Liste der verfügbaren »GDP's dar. Diese werden durch folgende Zahlen dargestellt: 1 - Balken 2 - Kreisbogen 3 - Tortenstück 4 - Kreis 5 - Ellipse 6 - elliptischer Bogen 7 - elliptisches Tortenstück 8 - Rechteck mit abgerundeten Ecken 9 - Rechteck, Ecken abgerundet mit Füllmuster 10 - Grafiktext (mit Randausgleich) Diese »GDP's sind auf dem Atari alle vorhanden.
work_out(25-34)	Fortlaufende Liste mit Attributen für die in 'work_out(15-24)' beschriebenen »GDP's: 0 = Linie 1 = Marker 2 = Text 3 = ausgefülltes Gebiet 4 = nichts
work_out(35)	Kann das Gerät Farbe ausgeben? 0 = nein (beim Atari mit monochromen Monitor) 1 = ja (beim Atari mit Farbmonitor)

work_out(36)	Ist das Drehen von Text möglich? 0 = nein 1 = ja (beim Atari, aber nur um 90 Grad)
work_out(37)	Können Flächen ausgefüllt werden? 0 = nein 1 = ja (beim Atari)
work_out(38)	Können Grafikoperationen mit Zellen durchgeführt werden? 0 = nein (beim Atari) 1 = ja (siehe »*v_cellarray()*)
work_out(39)	Anzahl der darstellbaren Farben 0 = mehr als 32767 2 = monochrom (schwarz und weiß) >2 = Anzahl der Farben Beim Atari von der eingestellten Auflösung abhängig
work_out(40)	Anzahl der Bildschirmsteuergeräte (zwei beim Atari) 1 = nur Tastatur 2 = Tastatur und anderes Steuergerät (Maus, Joystick etc.)
work_out(41)	Anzahl der Eingabegeräte 1 = Tastatur (beim Atari) 2 = noch ein anderes Gerät vorhanden
work_out(42)	Anzahl der Geräte mit Tastatur 1 = Funktionstasten vorhanden (beim Atari) 2 = Weiteres Gerät mit Tastatur verfügbar
work_out(43)	Anzahl der Geräte für Texteingabe 1 = Tastatur (beim Atari)
work_out(44)	Art der Workstation: 0 = reines Ausgabegerät (Drucker, Plotter etc) 1 = nur Eingabe 2 = sowohl Eingabe als auch Ausgabe (beim Atari) 3 = reserviert 4 = Metafileausgabe
work_out(45)	Kleinste Zeichenbreite (fünf beim Atari, monochrom)
work_out(46)	Kleinste Zeichenhöhe von der Basis- zur Toplinie (vier beim Atari, monochrom)
work_out(47)	Maximale Zeichenbreite (sieben beim Atari)
work_out(48)	Maximale Zeichenhöhe von der Basis- zur Toplinie (13 beim Atari, monochrom)
work_out(49)	Kleinste Linienbreite (eins beim Atari)
work_out(50)	= 0
work_out(51)	Maximale Linienbreite (40 beim Atari)
work_out(52)	= 0
work_out(53)	Kleinstmögliche Markerbreite (15 beim Atari)

```
work_out(54)        Kleinstmögliche Markerhöhe (elf beim Atari)
work_out(55)        Größte Markerbreite (120 beim Atari)
work_out(56)        Größte Markerhöhe (88 beim Atari)

                    Insbesondere bei den Werten von 'work_out'
                    45-56)', die für den Atari angegeben wur-
                    den, ist zu beachten, daß sich diese auf den
                    monochromen Bildschirm und »Rasterkoordinaten
                    beziehen!

                    Wem diese Informationen noch nicht reichen,
                    der kann sich mit »vq_extnd() noch weitere ho-
                    len.

contrl(2)    =  6
contrl(4)    =  45
contrl(6)    =  handle (Anmerkung: dies ist ein anderer Wert als
                       der an die Routine übergebene!)

intout(0)    =  work_out(0)
intout(1)    =  work_out(1)
.
.
.
intout(44)   =  work_out(44)

ptsout(0)    =  work_out(45)
ptsout(1)    =  work_out(46)
.
.
.
ptsout(11)   =  work_out(56)
```

V_opnwk

Zum »VDI gehörende Routine für Grafikausgabe

```
$01 = 1          v_opnwk(work_in, &handle, work_out)
word *                   work_in,  handle, work_out;
```

Diese Routine findet nur der Vollständigkeit halber Erwähnung! Denn bislang endete jeder Versuch, diese Routine aufzurufen, mit einem vollständigen Systemabsturz... (bei jeder uns bekannten Version des Betriebssystems!).

Abgesehen davon ist diese Routine auf dem Atari vollkommen überflüssig: das Öffnen der Arbeitsstation wird vom Betriebssystem vollständig übernommen, eine zweite Arbeitsstation läßt sich auf dem Atari leider (noch?) nicht öffnen. Dazu müßte ein weiterer

Grafiktreiber von Diskette nachgeladen werden. Grundvoraussetzung für die Verwendung der Routine ist im übrigen das Vorhandensein von "GDOS". Dies existiert als "GDOS.PRG" und muß bei der Verwendung verschiedener Programme (z. B. "GEMDRAW") vorher geladen werden. Es bleibt dann resident im Speicher.

Interessant an dieser Routine sind lediglich die Rückgabewerte, die man bekäme, funtionierte der Aufruf. Diese Werte bekommt man aber auch mithilfe von »*vq_extnd()* und »*v_opnvwk()*:

Im übrigen verweise ich auf »*v_opnvwk()*, mit der eine **virtuelle** Arbeitsstation geöffnet werden kann. Dort sind auch alle Rückgabeparameter beschrieben.

Eingabeparameter:

word *handle — Zeiger auf Rückgabevariable (s. u.)

word work_in() — Zeiger auf ein elf Worte umfassendes Array, in dem Werte für die Voreinstellung von Attributen stehen. Im folgenden die Bedeutung dieser elf Werte:

work_in(0) = Identifikationsnummer des Ausgabegeräts. Der in »ASSIGN.SYS stehende Gerätetreiber wird geladen.
work_in(1) = Linientyp (siehe »*vsl_type()*)
work_in(2) = Linienfarbe (siehe »*vsl_color()*)
work_in(3) = Markertyp (siehe »*vsm_type()*)
work_in(4) = Markerfarbe (siehe »*vsm_color()*)
work_in(5) = Zeichensatz (beim Atari geht momentan nur eine eins, außer, das neue »GDOS wurde gebootet)
work_in(6) = Textfarbe (siehe »*vst_color()*)
work_in(7) = Füllstil (siehe »*vsf_interior()*)
work_in(8) = Füllmuster (siehe »*vsf_style()*)
work_in(9) = Füllfarbe (siehe »*vsf_color()*)
work_in(10) = Dieses Flag gibt an, ob »normalisierte oder »Raster-Koordinaten verwendet werden sollen:
0 = Normalisierte Koordinaten
1 = Reserviert
2 = »Rasterkoordinaten

ACHTUNG: Normalisierte Koordinaten (»NDC) können nur dann verwendet werden, wenn »GDOS beim Start des Betriebssystems installiert worden ist! Siehe auch »GDOS!

Dann allerdings ist es auch möglich, die beiden Koordinatensystem gemischt zu verwenden, d. h. eine Workstation benutzt »NDC-Koordinaten, eine andere »Rasterkoordinaten.

word *handle
word *work_out Zeiger auf Variablen für Rückgabewerte (s. u.)

```
contrl(0)   =   1
contrl(1)   =   0
contrl(3)   =  11

intin(0)    =  work_in(0)
intin(1)    =  work_in(1)
.
.
.
intin(10)   =  work_in(10)
```

Ausgabeparameter:

word handle Kennummer der Arbeitsstation

word work_out(57) Ein 57 Worte umfassendes Array, in dem umfangreiche Angaben für die geöffnete virtuelle Arbeitsstation stehen. Die Bedeutung der Werte sind im Einzelnen (wobei die Parameter, die der Atari zurückgibt, in Klammern dahinter stehen):

work_out(0) Anzahl der addressierbaren »Pixel in X-Richtung. Ein Wert von 639 bedeutet, daß das Ausgabegerät »Pixel von 0 bis 639 setzen kann (beim Atari monochrom beispielsweise), also sind 640 »Pixel insgesamt vorhanden.

work_out(1) Anzahl der addressierbaren »Pixel in Y-Richtung.

work_out(2) Ist das Ausgabegerät in der Lage, Bilder in einer präzisen Skala auszugeben?
0 = ja (beim Atari)
1 = nein

work_out(3) Breite eines »Pixels in Micron (ein Micron = 1/1000 mm, 372 beim Atari)

work_out(4) Höhe eines »Pixels in Micron (ein Micron = 1/1000 mm, 372 beim Atari)

work_out(5) Anzahl der Schriftgößen (drei beim Atari), NULL bei kontinuierlicher Skala

work_out(6) Anzahl der Linientypen (sieben beim Atari)

work_out(7)	Anzahl der Linienbreiten, NULL bei kontinuierlicher Skala (wie etwa beim Atari)
work_out(8)	Anzahl der Markertypen (sechs beim Atari)
work_out(9)	Anzahl der Markergrößen, NULL bei kontinuierlicher Skala (acht beim Atari)
work_out(10)	Anzahl der unterstützten Zeichensätze (leider nur einen beim Atari)
work_out(11)	Anzahl der Musterarten (24 beim Atari)
work_out(12)	Anzahl der Schraffurarten (12 beim Atari)
work_out(13)	Anzahl der vordefinierten Farbe (beim Atari: zwei für monochrom)
work_out(14)	Anzahl der verfügbaren »Generalized Drawing Primitives (»GDP, zehn beim Atari)
work_out(15-24)	Die nächsten zehn Worte stellen eine Liste der verfügbaren »GDP's dar. Diese werden durch folgende Zahlen dargestellt: 1 - Balken 2 - Kreisbogen 3 - Tortenstück 4 - Kreis 5 - Ellipse 6 - elliptischer Bogen 7 - elliptisches Tortenstück 8 - Rechteck mit abgerundeten Ecken 9 - Rechteck, Ecken abgerundet mit Füllmuster 10 - Grafiktext (mit Randausgleich) Diese »GDP's sind auf dem Atari alle vorhanden.
work_out(25-34)	Fortlaufende Liste mit Attributen für die in 'work_out(15-24)' beschriebenen »GDP's: 0 = Linie 1 = Marker 2 = Text 3 = ausgefülltes Gebiet 4 = nichts
work_out(35)	Kann das Gerät Farbe ausgeben? 0 = nein (beim Atari mit monochromen Monitor) 1 = ja (beim Atari mit Farbmonitor)
work_out(36)	Ist das Drehen dvon Text möglich? 0 = nein 1 = ja (beim Atari)
work_out(37)	Können Flächen ausgefüllt werden? 0 = nein 1 = ja (beim Atari)
work_out(38)	Können Grafikoperationen mit Zellen durchgeführt werden? 0 = nein (beim Atari) 1 = ja (siehe »*v_cellarray()*)

work_out(39)	Anzahl der darstellbaren Farben 0 = mehr als 32767 2 = monochrom (schwarz und weiß) >2 = Anzahl der Farben Beim Atari von der eingestellten Auflösung abhängig
work_out(40)	Anzahl der Bildschirmsteuergeräte (zwei beim Atari) 1 = nur Tastatur 2 = Tastatur und anderes Steuergerät (Maus, Joystick etc.)
work_out(41)	Anzahl der Eingabegeräte 1 = Tastatur (beim Atari) 2 = noch ein anderes Gerät vorhanden
work_out(42)	Anzahl der Geräte mit Tastatur 1 = Funktionstasten vorhanden (beim Atari) 2 = Weiteres Gerät mit Tastatur verfügbar
work_out(43)	Anzahl der Geräte für Texteingabe 1 = Tastatur (beim Atari)
work_out(44)	Art der Workstation: 0 = reines Ausgabegerät (Drucker, Plotter etc) 1 = nur Eingabe 2 = sowohl Eingabe als auch Ausgabe (beim Atari) 3 = reserviert 4 = Metafileausgabe
work_out(45)	Kleinste Zeichenbreite (fünf beim Atari, monochrom)
work_out(46)	Kleinste Zeichenhöhe von der Basis- zur Toplinie (vier beim Atari, monochrom)
work_out(47)	Maximale Zeichenbreite (sieben beim Atari)
work_out(48)	Maximale Zeichenhöhe von der Basis- zur Toplinie (13 beim Atari, monochrom)
work_out(49)	Kleinste Linienbreite (eins beim Atari)
work_out(50)	= 0
work_out(51)	Maximale Linienbreite (40 beim Atari)
work_out(52)	= 0
work_out(53)	Kleinstmögliche Markerbreite (15 beim Atari)
work_out(54)	Kleinstmögliche Markerhöhe (elf beim Atari)
work_out(55)	Größte Markerbreite (120 beim Atari)
work_out(56)	Größte Markerhöhe (88 beim Atari)

Insbesondere bei den Werten von 'work_out' 45-56)', die für den Atari angegeben wurden, ist zu beachten, daß sich diese auf den monochromen Bildschirm und »Rasterkoordinaten beziehen!

Wem diese Informationen noch nicht reichen, der kann sich mit »*vq_extnd()* noch weitere holen.

```
contrl(2)    =  6
contrl(4)    = 45
contrl(6)    = handle

intout(0)    = work_out(0)
intout(1)    = work_out(1)
.
.
.
intout(44)   = work_out(44)

ptsout(0)    = work_out(45)
ptsout(1)    = work_out(46)
.
.
.
ptsout(11)   = work_out(56)
```

V_output_window

Zum »VDI gehörende Routine für Drucker

```
$05 = 5        v_output_window(handle, xyarray)
word                           handle;
word *                                 xyarray;
```

Mit dieser Funktion kann man einen rechteckigen Bildschirmausschnitt auf dem Drucker ausgeben.

Eingabeparameter:

word handle	Handle der »virtual Workstation für Ausgabe
word *xyarray	Zeiger auf ein vier Worte umfassendes Array, in dem sich die Koordinaten zweier diagonal gegenüberlieger Ecken des Rechtecks befinden: xyarray(0) = X-Koordinate der ersten Ecke xyarray(1) = Y-Koordinate der ersten Ecke xyarray(2) = X-Koordinate einer diagonal gegenüberliegenden Ecke xyarray(3) = Y-Koordinate einer diagonal gegenüberliegenden Ecke

```
contrl(0)   =  5
contrl(1)   =  2
contrl(3)   =  0
contrl(5)   = 21
contrl(6)   = handle
```

Ausgabeparameter:

```
contrl(2)   = 0
contrl(4)   = 0
```

V_pieslice

Zum »VDI gehörende Routine für Grafikausgabe

```
$0B = 11        v_pieslice(handle, x, y, radius, begang, endang)
word                       handle, x, y, radius, begang, endang;
```

Diese Routine zeichnet ein (mit einem Muster gefülltes) Tortenstück um den Punkt 'x', 'y' mit dem »Radius 'radius' vom Anfangswinkel 'begang' bis 'endang'. Der »Radius wird in »Pixeln angegeben. Nicht alle Ausgabegeräte beherrschen diese Funktion.

Die Ausgabe wird von folgenden Routinen beeinflußt:
- »*vsf_color()*
- »*vsf_interior()*
- »*vsf_perimeter()*
- »*vsf_style()*
- »*vswr_mode()*

Eingabeparameter:

```
word handle        Handle der »virtual Workstation für Ausgabe
word x             X-Koordinate des Mittelpunkts des Tortenstücks
word y             Y-Koordinate des Mittelpunkts des Tortenstücks
word radius        Radius des Tortenstücks
word begang        Anfangswinkel des Tortenstücks
word endang        Endwinkel des Tortenstücks
```

```
contrl(0)   = 11
contrl(1)   =  4
contrl(3)   =  2
contrl(5)   =  2

intin(0)    = begang
intin(1)    = endang
```

ptsin(0) = x
ptsin(1) = y
ptsin(6) = radius

Ausgabeparameter: keine

contrl(2) = 0
contrl(4) = 0

V_pline

Zum »VDI gehörende Routine für Grafikausgabe

```
$06 = 6          v_pline(handle, count, pxyarray)
word                     handle, count;
word *                                   pxyarray;
```

Diese Routine zeichnet eine Serie von miteinander verbundenen Linien. Dazu wird in dem Array 'pxyarray' eine Anzahl von X-Y-Koordinaten angegeben. Mindestens zwei dieser Koordinatenpaare müssen vorhanden sein. Sind diese nicht voneinander verschieden, so wird ein Punkt gezeichnet. Ist die mit »*vsl_width()* eingestellte Linienbreite größer als eins, so wird die Mitte der Linie auf die Koordinatenpaare gelegt.

Folgende Routinen beeinflussen die Ausgabe:
- »*vsl_color()*
- »*vsl_ends()*
- »*vsl_type()*
- »*vsl_width()*
- »*vswr_mode()*

Um Linien und Punkte zu zeichnen, können auch die (wesentlich schnelleren) »Line-A-Befehle genommen werden.

Eingabeparameter:

word handle	Handle der »virtual Workstation für Ausgabe
word count	Anzahl der Koordinatenpaare in 'pxyarray'. Die maximale Anzahl von Koordinatenpaaren kann über die Routine »*vq_extend()* abgefragt werden.

```
word *pxyarray      Adresse eines Arrays, welches die Koordinaten-
                    paare enthält. Die Größe (in Wörtern) ist
                    count * 2.
                    pxyarray(0)     = X1
                    pxyarray(1)     = Y1   1. Koordinatenpaar
                    .
                    .
                    .
                    pxyarray(2n-2) = Xn
                    pxyarray(2n-1) = Yn   n. Koordinatenpaar

contrl(0)    =  6
contrl(1)    =  n (= 'count')
contrl(3)    =  0
contrl(6)    =  handle

ptsin(0)     =  pxyarray(0)
ptsin(1)     =  pxyarray(1)
.
.
.
ptsin(2n-2)  =  pxyarray(2n-2)
ptsin(2n-1)  =  pxyarray(2n-1)
```

Ausgabeparameter:

```
contrl(2)    =  0
contrl(4)    =  0
```

V_pmarker

Zum »VDI gehörende Routine für Grafikausgabe

```
$07 = 7          v_pmarker(handle, count, pxyarray)
word                       handle, count;
word *                                     pxyarray;
```

Diese Routine setzt eine Anzahl von Markierungszeichen (= Marker) auf den Bildschirm. Der Typ des Markers wird mit »*vsm_type()* gesetzt, die Größe wird durch »*vsm_height()* und die Farbe durch »*vsm_color()* gesetzt.

Folgende Routinen beeinflussen die Ausgabe:
- »*vsm_color()*
- »*vsm_height()*
- »*vsm_type()*
- »*vswr_mode()*

Eingabeparameter:

word handle	Handle der »virtual Workstation für Ausgabe
word count	Anzahl der zu setzenden Marker
word *pxyarray	Zeiger auf ein 'count mal zwei' Worte umfassendes Array. In diesem Array stehen die Koordinaten der Marker (zuerst die X- dann die Y-Koordinate, 'count'-mal hintereinander).

```
contrl(0)    =  7
contrl(1)    =  count
contrl(3)    =  0
contrl(6)    =  handle

ptsin(0)     =  pxyarray(0)   X-Koordinate des ersten Markers
ptsin(1)     =  pxyarray(1)   Y-Koordinate des ersten Markers
ptsin(2)     =  pxyarray(2)   X-Koordinate des zweiten Markers
.
.
.
ptsin(2n-2)  =  pxyarray(2*count-2)
ptsin(2n-1)  =  pxyarray(2*count-1)
```

Ausgabeparameter:

```
contrl(2)    =  0
contrl(4)    =  0
```

V_rfbox

Zum »VDI gehörende Routine für Grafikausgabe

```
$0B = 11        v_rfbox(handle, xyarray)
word                    handle;
word *                          xyarray;
```

Diese Routine zeichnet ein Rechteck mit abgerundeten Ecken, welches mit dem gerade aktuellen Füllmuster gefüllt wird.

Die Ausgabe wird von folgenden Routinen beeinflußt:
- »*vsf_color()*
- »*vsf_interior()*
- »*vsf_perimeter()*
- »*vsf_style()*
- »*vswr_mode()*

Eingabeparameter:

word handle — Handle der »virtual Workstation für Ausgabe

word *xyarray — Zeiger auf ein Array von Worten mit den Werten von zwei diagonal gegenüberliegenden Ecken des Rechtecks:

```
xyarray[0] = X1-Koordinate
xyarray[1] = Y1-Koordinate
xyarray[2] = X2-Koordinate
xyarray[3] = Y2-Koordinate
```

```
contrl(0)   =  11
contrl(1)   =   2
contrl(3)   =   0
contrl(5)   =   9
contrl(6)   =  handle

ptsin(0)    =  X1  = xyarray[0]
ptsin(1)    =  Y1  = xyarray[1]
ptsin(2)    =  X2  = xyarray[2]
ptsin(3)    =  Y2  = xyarray[3]
```

Ausgabeparameter: keine

```
contrl(2)   =  0
contrl(4)   =  0
```

V_rbox

Zum »VDI gehörende Routine für Grafikausgabe

```
$0B = 11        v_rbox(handle, xyarray)
word                  handle;
word *                         xyarray;
```

Diese Routine zeichnet ein Rechteck mit abgerundeten Ecken. Das Rechteck wird nicht ausgefüllt.

Die Ausgabe wird von folgenden Routinen beeinflußt:
- »*vsl_color()*
- »*vsl_ends()*
- »*vsl_type()*
- »*vsl_width()*
- »*vswr_mode()*

Eingabeparameter:

word handle Handle der »virtual Workstation für Ausgabe

word *xyarray Zeiger auf ein Array von Worten mit den Werten von zwei diagonal gegenüberliegenden Ecken des Rechtecks:

```
xyarray[0] = X1-Koordinate
xyarray[1] = Y1-Koordinate
xyarray[2] = X2-Koordinate
xyarray[3] = Y2-Koordinate
```

```
contrl(0)   =  11
contrl(1)   =   2
contrl(3)   =   0
contrl(5)   =   8
contrl(6)   =  handle

ptsin(0)    =  X1  = xyarray[0]
ptsin(1)    =  Y1  = xyarray[1]
ptsin(2)    =  X2  = xyarray[2]
ptsin(3)    =  Y2  = xyarray[3]
```

Ausgabeparameter:

```
contrl(2)   =  0
contrl(4)   =  0
```

V_rmcur

Zu VDI gehörende Escape-Funktion

```
$05 = 5          v_rmcur(handle)
word                     handle;
```

Diese Funktion ist das Gegenstück zu »*v_dspcur()*. Der damit gezeichnete Cursor wird wieder entfernt.

Eingabeparameter:

word handle Handle der »virtual Workstation für Ausgabe

```
contrl(0)   =   5
contrl(1)   =   0
contrl(3)   =   0
contrl(5)   =  19
contrl(6)   =  handle
```

Ausgabeparameter:

```
contrl(2)   =  0
contrl(4)   =  0
```

v_show_c

Zum »VDI gehörende Routine für Grafikausgabe

```
$7A = 122        v_show_c(handle, reset)
word                      handle, reset;
```

Diese Routine macht den Mauscursor wieder sichtbar. Damit bewirkt sie dasselbe wie »*graf_mouse()*. Sehen Sie bitte unter dem Stichwort nach!

Aber: wenn 'reset' gleich NULL ist, so wird der Mauscursor angezeigt, egal wie oft vorher der Mauszeiger versteckt wurde!

Eingabeparameter:

```
word handle        Handle der »virtual Workstation für Ausgabe

word reset         = 0: Mauscursor unbedingt anzeigen
                   = 1: Mauscursor anzeigen, wenn die Anzahl der
                        der Befehle zum Unsichtbarmachen ausge-
                        glichen ist.

contrl(0)   =  122
contrl(1)   =    0
contrl(3)   =    1
contrl(6)   =  handle

intin(0)    =  reset
```

Ausgabeparameter:

```
contrl(2)   =  0
contrl(4)   =  0
```

v_rvoff

Zum »VDI gehörende Escape-Routine für Text

```
$05 = 5          v_rvoff(handle)
word                     handle;
```

Der Aufruf dieser Routine bewirkt, daß jeder im folgenden ausge-

gebene Text wieder normal dargestellt wird (schwarz auf weißem Grund).

Siehe auch »Bildschirmsteuerung.

Eingabeparameter:

word handle Handle der »virtual Workstation für Ausgabe

```
contrl(0)   =  5
contrl(1)   =  0
contrl(3)   =  0
contrl(5)   =  14
contrl(6)   =  handle
```

Ausgabeparameter: keine

```
contrl(2)   =  0
contrl(4)   =  0
```

V_rvon

Zum »VDI gehörende Escape-Routine für Text

```
$05 = 5        v_rvon(handle)
word                 handle;
```

Der Aufruf dieser Routine bewirkt, daß jeder im folgenden ausgegebene Text invertiert dargestellt wird (weiß auf schwarzem Grund).

Siehe auch »Bildschirmsteuerung.

Eingabeparameter:

word handle Handle der »virtual Workstation für Ausgabe

```
contrl(0)   =  5
contrl(1)   =  0
contrl(3)   =  0
contrl(5)   =  13
contrl(6)   =  handle
```

Ausgabeparameter:

```
contrl(2)   =  0
contrl(4)   =  0
```

V_updwk

Zum »VDI gehörende Routine

```
$04 = 4          v_updwk(handle)
word                     handle;
```

Mit dieser Routine wird auf dem Atari **nichts** bewirkt. Sie macht nur dann einen Sinn, wenn es sich bei der Arbeitsstation um einen Drucker oder Plotter handelt (Ausgabe des Puffers), aber die Bildschirmausgabe ist ungepuffert. Vielleicht wird es ja doch eines Tages möglich, eine eigene Workstation zu öffnen, und dann... (siehe auch »*v_opnwk()*).

Eingabeparameter:

```
word handle        Handle der »virtual Workstation für Ausgabe

contrl(0)   =  4
contrl(1)   =  0
contrl(3)   =  0
contrl(6)   =  handle
```

Ausgabeparameter:

```
contrl(2)   =  0
contrl(4)   =  0
```

Vektor

Siehe »Exception.

Verify

Engl.: Überprüfung.

Vex_butv

Zum »VDI gehörende Routine für Grafikausgabe

```
$7D = 125        vex_butv(handle, pusrcode, psavcode)
word                     handle;
long                              pusrcode, psavcode;
```

Mit dieser Routine kann ihr Programm jedesmal eine Aktion ausführen, wenn sich der Zustand eines der Mausknöpfe verändert.

Dazu muß die Adresse einer eigenen Routine angegeben werden. Die eigene Routine wird mit einem JSR-Befehl (Jump to Subroutine) angesprungen. D0.W enthält den Zustand der Mausknöpfe (in derselben

Art kodiert wie bei »*vq_mouse()*). Bevor die eigene Routine ein RTS ausführt, kann der Wert in D0.W den eigenen Bedürfnissen angepaßt werden.

Die eigene Routine muß alle Register, die sie verwendet, vorher wieder auf den alten Stand bringen. Am Ende der eigenen Routine muß ein RTS (Return from Subroutine) stehen.

Wenn die eigene Routine aufgerufen wird, so sind alle Interrupts abgeschaltet. Sie sollten auch nicht wieder eingeschaltet werden.

Eingabeparameter:

word handle	Handle der »virtual Workstation für Ausgabe
long pusrcode	Adresse der eigenen Routine
long psavcode	Adresse eines Rückgabeparameters (s. u.)

```
contrl(0)   =  125
contrl(1)   =    0
contrl(3)   =    0
contrl(6)   =  handle
contrl(7-8) =  pusrcode
```

Ausgabeparameter:

long psavcode	Enthält hinterher die Adresse der alten »VDI-Routine, die den Zustand der Mausknöpfe speichert (für späteren Gebrauch durch andere Routinen)

```
contrl(2)   =  0
contrl(4)   =  0
contrl(9-10)=  psavcode
```

Vex_curv

Zum »VDI gehörende Routine für Grafikausgabe

```
$7F = 127       vex_curv(handle, pusrcode, psavcode)
word                     handle;
long                              pusrcode, psavcode;
```

Mit dieser Routine kann eine Aktion ausgeführt werden, und zwar jedesmal dann, bevor der Mauscursor gezeichnet wird oder auch anstatt der »VDI-Routine, die den Mauscursor zeichnet.

Der »Mauszeiger wird jedesmal dann neu gezeichnet, wenn der Benutzer die Maus bewegt.

Dazu muß die Adresse einer eigenen Routine angegeben werden. Die eigene Routine wird mit einem JSR-Befehl (Jump to Subroutine) angesprungen. Register D0.W enthält die X-Position, Register D1.W enthält die Y-Position der Maus. Falls die eigene Routine den Cursor nicht selbst neu zeichnet, so sollte die alte Routine mit einem JSR-Befehl aufgerufen werden (Diese zeichnet den Mauszeiger). Register D0.W und D1.W müssen die Maus-Koordinaten enthalten. Die eigene Routine muß alle Register, die sie verwendet, vorher wieder auf den alten Stand bringen. Am Ende der eigenen Routine muß ein RTS (Return from Subroutine) stehen.

Wenn die eigene Routine aufgerufen wird, so sind alle Interrupts abgeschaltet. Sie sollten auch nicht wieder eingeschaltet werden.

Eingabeparameter:

```
word handle        Handle der »virtual Workstation für Ausgabe

long pusrcode      Adresse der eigenen Routine

long psavcode      Adresse eines Rückgabeparameters (s. u.)

contrl(0)    =  127
contrl(1)    =    0
contrl(3)    =    0
contrl(6)    =  handle
contrl(7-8)  =  pusrcode
```

Ausgabeparameter:

```
long psavcode      Enthält hinterher die Adresse der alten
                   »VDI-Routine, die den Mauscursor zeichnet

contrl(2)    =  0
contrl(4)    =  0
contrl(9-10)=  psavcode
```

Vex_motv

Zum »VDI gehörende Routine für Grafikausgabe

```
$7E = 126      vex_motv(handle, pusrcode, psavcode)
word                    handle;
long                            pusrcode, psavcode;
```

Mit dieser Routine kann ihr Programm jedesmal eine Aktion ausfüh-

ren, bevor der »Mauszeiger an eine neu Position bewegt wird. Die eigene Routine, wird angesprungen, **nachdem** die neuen X-Y-Koordinaten des Mauscursors berechnet wurden, aber **bevor** die neue Mausposition gespeichert wird und bevor der »Mauszeiger neu gezeichnet wird.

Dazu muß die Adresse einer eigenen Routine angegeben werden. Die eigene Routine wird mit einem JSR-Befehl (Jump to Subroutine) angesprungen. Register D0.W enthält die X-Position, Register D1.W enthält die Y-Position der Maus. Dies muß auch der Fall sein, wenn die eigene Routine wieder verlassen wird, aber es kann sich natürlich auch um geänderte Werte handeln... So ist es zum Beispiel möglich, einen Bereich zu erzeugen, den die Maus nicht verlassen kann.

Die eigene Routine muß alle Register, die sie verwendet, vorher wieder auf den alten Stand bringen. Am Ende der eigenen Routine muß ein RTS (Return from Subroutine) stehen.

Wenn die eigene Routine aufgerufen wird, so sind alle Interrupts abgeschaltet. Sie sollten auch nicht wieder eingeschaltet werden.

Eingabeparameter:

word handle	Handle der »virtual Workstation für Ausgabe
long pusrcode	Adresse der eigenen Routine
long psavcode	Adresse eines Rückgabeparameters (s. u.)

```
contrl(0)   =  126
contrl(1)   =    0
contrl(3)   =    0
contrl(6)   =  handle
contrl(7-8) =  pusrcode
```

Ausgabeparameter:

long psavcode	Enthält hinterher die Adresse der alten »VDI-Routine, die die Mausbewegung bearbeitet

```
contrl(2)   =  0
contrl(4)   =  0
contrl(9-10)=  psavcode
```

Vex_timv

Zum »VDI gehörende Routine

```
$76 = 118        vex_timv(handle, tim_addr, otim_addr, &tim_conv)
word                     handle;
long                             tim_addr, otim_addr;
word *                                                 tim_conv;
```

Diese Routine ermöglicht es Ihnen, bei jedem Auftreten des Timer--Interrupts einen eigenen Programmteil auszuführen. Der Timer-Interrupt tritt 200 mal pro Sekunde auf. Ihre Routine wird mit einem JSR aufgerufen und muß mit einem RTS enden.

Die eigene Routine muß alle Register, die sie verwendet, vorher wieder auf den alten Stand bringen. Am Ende der eigenen Routine muß ein RTS (Return from Subroutine) stehen.

Wenn die eigene Routine aufgerufen wird, so sind alle Interrupts abgeschaltet. Sie sollten auch nicht wieder eingeschaltet werden.

Eingabeparameter:

```
word handle        Handle der »virtual Workstation für Ausgabe

long tim_addr      Adresse der eigenen Routine

long otim_addr
word *tim_conv     Adressen von Rückgabeparametern (s. u.)

contrl(0)   =  118
contrl(1)   =    0
contrl(3)   =    0
contrl(6)   =  handle
contrl(7-8) =  tim_addr
```

Ausgabeparameter:

```
long otim_addr     Enthält hinterher die Adresse der alten
                   Timer-Routine

word tim_conv      Anzahl der Millisekunden pro Timer-Interrupt

contrl(2)   =  0
contrl(4)   =  0
contrl(9-10)=  tim_conv

intout(0)   =  tim_conv
```

Video-Speicher

Siehe »Bildschirmspeicher

Virtual workstation

Engl.: virtuelle Arbeitsstation, zu deutsch: virtuelle Arbeitsstation.

Auf dem »Bildschirm des Atari können mehrere Workstations gleichzeitig geöffnet werden (siehe »*v_opnvwk()*). Jede dieser Workstation kann wie ein eigener »Bildschirm betrachtet werden (auch wenn jede Workstation Zugriff auf den ganzen »Bildschirm hat). Jede »VDI-Funktion kann nun auf jede geöffnete Arbeitsstation etwas ausgeben. Da sich für jede dieser virtuellen Bildschirme die Attribute (z. B. Linienbreite) gesondert einstellen lassen, zeichnet die Funktion »*v_pline()* jeweils auf den verschiedenen Workstations eine verschieden breite Linie (eben je nach Einstellung).

Der Zugriff auf die verschiedenen Arbeitsstationen erfolgt über einen "handle", d. h. jede Workstation erhält eine andere Nummer. Fast alle »VDI-Funktionen benötigen als ersten Parameter diesen "handle", um die korrekte Workstation ansprechen zu können.

Virtuelle Arbeitsstation

siehe »virtual Workstation und »*v_opnvwk()*

Vm_filename

Zum »VDI gehörende Escape-Routine für Metafiles

```
$05 = 5          vm_filename(handle, filename)
word                         handle;
byte *                                filename;
```

Wenn ein »Metafile erzeugt wird, so erhält es den Standard-Namen "GEMFILE.GEM". Falls man einen anderen Namen haben möchte, so muß man diese Routine verwenden.

Eingabeparameter:

word handle	Handle der »virtual Workstation für Ausgabe
byte *filename	Zeiger auf den neuen Namen für das Metafile. Der Namen kann den kompletten Suchpfad inklusive Ordner etc. enthalten.

```
contrl(0)    =    5
contrl(1)    =    0
contrl(3)    =    Länge des Dateinamens
contrl(5)    =  100
contrl(6)    =  handle

intin(0)     =  1. Zeichen des Dateinamens
intin(1)     =  2. Zeichen des Dateinamens
.
.
.
intin(n)     =  Letztes Zeichen des Dateinamens (= Null)
```

Ausgabeparameter: keine

```
contrl(2)    =  0
contrl(4)    =  0
```

Vq_cellarry

Zum »VDI gehörende Routine für Grafikausgabe

```
$1B = 27        vq_cellarray(handle, pxyarray, row_length,
word                         handle,           row_length;
word *                               pxyarray;
                             num_rows, &el_used, &rows_used,
word                         num_rows;
word *                                    el_used,  rows_used;
                             &status, colarray)
word *                        status, colarray;
```

Diese Routine würde benötigt, falls »*v_cellarray()* funktionierte. Dann könnte man nämlich Informationen über die Zellen des Feldes abfragen, das man mit »*v_cellarray()* kreiert hat. Aber so macht diese Funktion keinen Sinn.

Vq_chcells

Zum »VDI gehörende Escape-Routine für Text

```
$05 = 5         v_chcells(handle, &rows, &columns)
word                      handle;
word *                              rows,  columns;
```

Mit dieser Routine kann abgefragt werden, wo der Cursor überall auf dem (Text-) Bildschirm positioniert werden kann. Nicht alle Geräte beherrschen die Cursor-Positionierung.

Eingabeparameter:

word handle — Handle der »virtual Workstation für Ausgabe

word *rows
word *columns — Zeiger auf die Rückgabewerte (s. u.)

```
contrl(0)   =  5
contrl(1)   =  0
contrl(3)   =  0
contrl(5)   =  1
contrl(6)   =  handle
```

Ausgabeparameter:

word rows — Anzahl der Zeilen, in denen der Textcursor positioniert werden kann.

word columns — Anzahl der Spalten, in denen der Textcursor positioniert werden kann.

```
contrl(2)   =  0
contrl(4)   =  2

intout(0)   =  rows
intout(1)   =  columns
```

Vq_color

Zum »VDI gehörende Routine für Grafikausgabe

```
$1A = 26        vq_color(handle, color_index, set_flag, rgb)
word                 handle, color_index, set_flag
word *                                            rgb;
```

Mit dieser Routine können Sie die eingestellten Farbwerte abfragen. Es sind insgesamt 16 Farben einstellbar (niedrige Auflösung, wählbar aus 512) . Allerdings werden auf einem monochromen Ausgabegerät alle Farbeinstellungen als "Weiß" angesehen. Der Farbindex Null ist die Hintergrundfarbe.

Jede Farbe besteht aus einer Mischung von Rot, Grün und Blau ("RGB"). Der Anteil dieser Grundfarben wird mit Werten von 0 bis 1000 angegeben.

Es können die eingestellten Farben oder aber die gewünschten abgefragt werden.

Mit »*vs_color()* können die Farbwerte (Farbanteile) der Farben eingestellt werden.

Eingabeparameter:

word handle	Handle der »virtual Workstation für Ausgabe
word color_index	Farbindex
word set_flag	Mit diesem Flag kann eingestellt werden, ob man die realisierten Farbanteile zurückbekommt (set_flag = 1) oder die gewünschten (set_flag = 0)
word *rgb	Zeiger auf ein Array aus drei Wörtern: rgb_in(0) = Rotanteil (von 0 bis 1000) rgb_in(1) = Grünanteil -"- rgb_in(2) = Blauanteil -"-

```
contrl(0)   =  26
contrl(1)   =   0
contrl(3)   =   2
contrl(6)   =  handle

intin(0)    =  color_index
intin(1)    =  set_flag
```

Ausgabeparameter:

```
contrl(2)   =  0
contrl(4)   =  4

intout(0)   =  color_index
intout(1)   =  rgb(0)
intout(2)   =  rgb(1)
intout(3)   =  rgb(2)
```

Vq_extnd

Zum »VDI gehörende Routine für Grafikausgabe

```
$66 = 102      vq_extnd(handle, owflag, work_out)
word                 handle, ow_flag;
word *                          work_out;
```

Mit dieser Routine lassen sich Informationen über das System abfragen. Dies können entweder dieselben sein wie bei »*v_opnwk()* oder aber weitergehende. Da »GEM nicht für den Atari entwickelt

wurde, ist diese Routine vor allem bei Portierungen von Programmen wichtig.

Eingabeparameter:

word handle — Handle der »virtual Workstation für Ausgabe

word owflag — Dieses Flag gibt an, welcher Art die abgefragten Informationen sein sollen:
'owflag' = 0 ergibt dieselben Informationen wie bei »*v_opnwk()*
'owflag' = 1 gibt die erweiterten Informationen zurück
An dieser Stelle werden nur die erweiterten Informationen beschrieben.

word *work_out — Zeiger auf ein 56 Worte umfassendes Array, in dem die Informationen zurückgegeben werden.

```
contrl(0)   =  102
contrl(1)   =    0
contrl(3)   =    1
contrl(6)   =  handle

intin(0)    =  owflag
```

Ausgabeparameter:

word work_out(45) — Falls 'owflag' = 1 stehen in diesem Array die im folgenden näher beschriebenen Werte. Falls Sie die Bedeutung der Informationen bei 'owflag' = 0 wissen wollen, schauen Sie bitte bei »*v_opnwk()* nach.

word work_out(0) — Art des Bildschirms:
0 = kein Bildschirm (z. B. Drucker oder Plotter)
1 = Getrennte Bildschirme für Grafik- und Textmodus mit separatem Controller
2 = Separate Controller für Grafik- und Textmodus, aber für beide den gleichen Bildschirm
3 = Gleicher Controller für Text- und Grafikmodus, aber verschiedene Videospeicher
4 = Gleicher Controller für Text- und Grafikspeicher und gleicher Videospeicher
Beim Atari bekommt man den Wert vier zurück

word work_out(1)	Anzahl der möglichen Hintergrundfarben (beim Atari eine)
word work_out(2)	Anzahl der vorhandenen Texteffekte (siehe »*vst_effects()*), beim Atari sind das 31 (fünf verschiedene Texteffekte in 31 Kombinationen)
word work_out(3)	Vergrößerung bei Rasteroperationen möglich: 0 = nein (Atari) 1 = ja
word work_out(4)	Anzahl der Bildebenen bei Farbe (Atari: 1 - 4)
word work_out(5)	Farbtabelle 0 = nein (Atari) 1 = ja
word work_out(6)	Anzahl der 16x16 »Pixel großen Rasteroperationen pro Sekunde (beim Atari immerhin 1000!)
word work_out(7)	Fülloperation bei beliebiger Fläche möglich? 0 = nein 1 = ja (Atari)
word work_out(8)	Text-Rotation (siehe »*vst_rotation()*) möglich? 0 = nein 1 = ja, aber nur in 90-Grad-Winkeln (Atari) 2 = Rotation in beliebigem Winkel
word work_out(9)	Anzahl der Schreibmodi (beim Atari vier)
word work_out(10)	Mögliche Eingabemodi: 0 = keine Eingabe 1 = nur request 2 = request und sample (Atari)
word work_out(11)	Kann der Text beliebig positioniert werden? 0 = nein 1 = ja (Atari)
word work_out(12)	Farbwechsel bei Plottern oder Druckern 0 = nein (Atari) 1 = ja
word work_out(13)	Löschband bei Druckern 0 = nein (Atari) 1 = bei Linien 2 = auch bei Rechtecken

```
word work_out(14)  Maximale Anzahl der Punkte bei Linien, Markern
                   und ausgefüllten Flächen (beim Atari 128)

word work_out(15)  Maximale Größe des intin-Arrays
                   -1 = keine Beschränkung (Atari)

word work_out(16)  Anzahl der Maustasten (beim Atari zwei)

word work_out(17)  Kann der Typ der breiten Linien festgelegt
                   werden?
                   0 = nein (Atari)
                   1 = ja

word work_out(18)  Schreibmodi für breite Linien (Atari NULL)

word work_out(19)  reserviert, Null
.                        -"-
.                        -"-
.                        -"-
word work_out(44)  reserviert, Null

contrl(2)    =   6
contrl(4)    =  45

intout(0)    =  word work_out(0)
.
.
.
intout(44)   =  word work_out(44)

ptsout(0)    =  word work_out(45) = 0
.
.
.
ptsout(11)   =  word work_out(56) = 0
```

Vq_key_s

Zum »VDI gehörende Routine

```
$80 = 128        vq_key_s(handle, &pstatus)
word                     handle;
word *                           pstatus;
```

Mit dieser Routine können Sie den Status (Zustand) der Shift-Control- und Alternate-Taste abfragen.

Eingabeparameter:

word handle — Handle der »virtual Workstation für Ausgabe

word *pstatus — Zeiger auf Wort für Rückgabewert (s. u.)

```
contrl(0)   =  128
contrl(1)   =    0
contrl(3)   =    0
contrl(6)   =  handle
```

Ausgabeparameter:

word pstatus — Enthält (bitweise kodiert) den Status der Shift- CTRL- und ALT-Taste. Wenn das entsprechende Bit gesetzt ist, so ist die zugehörige Taste zur Zeit der Abfrage gedrückt.

Bit	Taste
0	Rechte Shift-Taste
1	Linke Shift-Taste
2	Control-Taste
3	Alternate-Taste

```
contrl(2)   =  0
contrl(4)   =  1

intout(0)   =  pstatus
```

Vq_mouse

Zum »VDI gehörende Routine für Grafikausgabe

```
$7C = 124      vq_mouse(handle, &pstatus, &x, &y)
word                    handle;
word *                           pstatus,  x,  y;
```

Dies ist die »VDI-Routine, um Informationen über den Zustand der Maus zu bekommen (X-Y-Position, welche Mausknöpfe gedrückt wurden).

Eingabeparameter:

word handle — Handle der »virtual Workstation für Ausgabe

word *status
word *x
word *y — Zeiger auf Variablen für Rückgabewerte (s. u.)

```
contrl(0)   =  124
contrl(1)   =    0 contrl(3)
  =    0 contrl(6)
  =  handle
```

Ausgabeparameter:

word pstatus	Zustand der Mausknöpfe. Der linke Mausknopf ist gedrückt, wenn Bit Null in 'pstatus' gesetzt ist; ist Bit Eins gesetzt, so wurde der rechte Mausknopf gedrückt.
word x	Die X-Koordinate des Mauszeigers
word y	Die Y-Koordinate des Mauszeigers

```
contrl(2)   =  1
contrl(4)   =  1

intout(0)   =  pstatus

ptsout(0)   =  x
ptsout(1)   =  y
```

Vq_tabstatus

Zum »VDI gehörende Routine

```
$05 = 5          status = vq_tabstatus(handle)
word                                    handle;
```

Diese Escape-Funktion gibt die Verfügbarkeit eines Grafik-Eingabegerätes an, wie etwa »Maus, Joystick, Grafik-Tablett oder etwas ähnlichem.

Eingabeparameter:

word handle	Handle der »virtual Workstation für Ausgabe

```
contrl(0)   =   5
contrl(1)   =   0
contrl(3)   =   0
contrl(5)   =  16
contrl(6)   =  handle
```

Ausgabeparameter:

status — Status des Grafik-Tabletts:
0 = nicht verfügbar
1 = verfügbar

contrl(2) = 0
contrl(4) = 1

intout(0) = status

Vqf_attributes

Zum »VDI gehörende Routine für Grafikausgabe

```
$25 = 37        vqf_attributes(handle, attrib)
word                           handle;
word *                                 attrib;
```

Hiermit können die Attribute abgefragt werden, die das Aussehen des aktuellen Füllmusters bestimmen. Diese sind:
-Füllfarbe (gesetzt mit »*vsf_color()*)
-Füllstil (gesetzt mit »*vsf_interior()*)
-Füllmuster (gesetzt mit »*vsf_style()*)
-Schreibmodus (gesetzt mit »*vswr_mode()*)

Für eine genauere Beschreibung sehen Sie bitte unter den Routinen nach, mit denen die Attribute gesetzt werden!

Eingabeparameter:

word handle — Handle der »virtual Workstation für Ausgabe

word *attrib — Zeiger auf ein fünf Worte umfassendes Array, in dem die gesetzten Attribute zurückgegeben werden (s. u.).

contrl(0) = 37
contrl(1) = 0
contrl(3) = 0
contrl(6) = handle

Ausgabeparameter:

word attrib(0) — Füllstil (siehe »*vsf_interior()*)

word attrib(1) — Füllfarbe (siehe »*vsf_color()*)

word attrib(2) — Füllmuster(siehe »*vsf_style()*)

word attrib(3) Schreibmodus (siehe »*vswr_mode()*)

word attrib(4) Sichtbarkeit des Füllrandes (siehe »*vsf_perimeter()*)

```
contrl(2)   =  0
contrl(4)   =  5

intout(0)   =  word attrib(0)
intout(1)   =  word attrib(1)
intout(2)   =  word attrib(2)
intout(3)   =  word attrib(3)
intout(4)   =  word attrib(4)
```

Vqin_mode

Zum »VDI gehörende Routine

```
$73 = 115      vqin_mode(handle, dev_type, &input_mode)
word                 handle, dev_type;
word *                                   input_mode;
```

Für ein logisches Eingabegerät kann hiermit abgefragt werden, ob es im »Request- oder »Sample-Modus arbeitet.

Eingabeparameter:

word handle Handle der »virtual Workstation für Ausgabe

word dev_type Art des logischen Eingabegeräts:

1 = Locator	Gerät, das geeignet ist, Positionen einzugeben (z. B. »Maus)
2 = Valuator	Eingaben, die einen Wert verändern
3 = Choice	Auswahlgerät, beim Atari sind das die Funktionstasten
4 = String	Gerät zum Eingeben einer Zeichenkette

word *input_mode Zeiger auf Variable für Rückgabewert (s. u.)

```
contrl(0)   =  115
contrl(1)   =    0
contrl(3)   =    1
contrl(6)   =  handle

intin(0)    =  dev_type
```

Ausgabeparameter:

word input_mode Eingabemodus:
1 = »Request
2 = »Sample

contrl(2) = 0
contrl(4) = 1

intout(0) = input_mode

Vql_attributes

Zum »VDI gehörende Routine für Grafikausgabe

$23 = 35 vql_attributes(handle, attrib)
word handle;
word * attrib;

Mit dieser Routine kann man die eingestellten Werte (Attribute) abfragen, die das Zeichnen der Linien beeinflussen. Dies sind:
- Linientyp
- Linienfarbe
- Linienbreite
- Aussehen der Enden
- Schreibmodus

Eingabeparameter:

word handle Handle der »virtual Workstation für Ausgabe

word *attrib Zeiger auf ein vier Worte umfassendes Array, in dem die Rückgabewerte stehen (s. u.)

contrl(0) = 35
contrl(1) = 0
contrl(3) = 0
contrl(6) = handle

Ausgabeparameter:

word attrib(0) Linientyp (siehe »*vsl_type()*)

word attrib(1) Linienfarbe (siehe »*vsl_color()*)

word attrib(2) Schreibmodus (siehe »*vswr_mode()*)

word attrib(3) Breite der Linien (im augenblicklichen Koordinatensystem)

```
contrl(2)   =  1
contrl(4)   =  5

intout(0)   =  attrib(0)
intout(1)   =  attrib(1)
intout(2)   =  attrib(2)

ptsout(0)   =  attrib(3)
ptsout(1)   =  0
```

Vqm_attributes

Zum »VDI gehörende Routine für Grafikausgabe

```
$24 = 36        vqm_attributes(handle, attrib)
word                    handle;
word *                          attrib;
```

Mit dieser Routine kann man die eingestellten Werte (Attribute) abfragen, die das Zeichnen der Marker beeinflussen. Dies sind:
- Markertyp
- Markerfarbe
- Markerhöhe
- Markerbreite
- Schreibmodus

Eingabeparameter:

word handle	Handle der »virtual Workstation für Ausgabe
word *attrib	Zeiger auf ein vier Worte umfassendes Array, in dem die Rückgabewerte stehen (s. u.)

```
contrl(0)   =  36
contrl(1)   =   0
contrl(3)   =   0
contrl(6)   =  handle
```

Ausgabeparameter:

word attrib(0)	Markertyp (siehe »*vsm_type()*)
word attrib(1)	Markerfarbe (siehe »*vsm_color()*)
word attrib(2)	Schreibmodus (siehe »*vswr_mode()*)
word attrib(3)	Breite des Markers (im augenblicklichen Koordinatensystem)

```
contrl(2)   =  1
contrl(4)   =  3

intout(0)   =  attrib(0)
intout(1)   =  attrib(1)
intout(2)   =  attrib(2)

ptsout(0)   =  attrib(3)
ptsout(1)   =  Höhe des Markers
```

Vqt_attributes

Zum »VDI gehörende Routine für Grafikausgabe

```
$26 = 38        vqt_attributes(handle, attrib)
word                            handle;
word *                                  attrib;
```

Mithilfe dieser Routine können Sie Informationen über alle Attribute der Grafik-Textausgabe erfahren:

- Textgröße (gesetzt mit »*vst_height()* oder »*vst_point()*)
- Textfarbe (gesetzt mit »*vst_color()*)
- Textausrichtung (gesetzt mit »*vst_allignment()*)
- Textwinkel (gesetzt mit »*vst_rotation()*)
- Schreibmodus (gesetzt mit »*vswr_mode()*)

Die Bedeutung der einzelnen Attribute erfahren Sie bei den Routinen, die die Attribute setzen.

Eingabeparameter:

word handle	Handle der »virtual Workstation für Ausgabe
word *attrib	Zeiger auf ein zehn Worte umfassendes Array, in das die Rückgabewerte hineingeschrieben werden sollen

```
contrl(0)   =  38
contrl(1)   =   0
contrl(3)   =   0
contrl(6)   =  handle
```

Ausgabeparameter:

word attrib(0)	Gültiger Zeichensatz
word attrib(1)	Textfarbe

word attrib(2)	Winkel der Textausgabe in Zehntelgraden (also 0, 900, 1800 oder 2700 auf dem Atari)
word attrib(3)	Horizontale Textausrichtung (siehe »*vst_alignment()*)
word attrib(4)	Vertikale Textausrichtung (siehe »*vst_alignment()*)
word attrib(5)	Schreibmodus
word attrib(6)	Breite des breitesten Zeichens im Zeichensatz (im gerade gültigen Koordinatensystem)
word attrib(7)	Höhe des breitesten Zeichens im Zeichensatz (im gerade gültigen Koordinatensystem)
word attrib(8)	Breite des Kastens im Zeichensatz, der das breiteste Zeichen inklusive Rand umschließen kann (im gerade gültigen Koordinatensystem)
word attrib(9)	Höhe des Kastens im Zeichensatz, der das höchste Zeichen inklusive Rand umschließen kann (im gerade gültigen Koordinatensystem)

```
contrl(2)   =  2
contrl(4)   =  6

intout(0)   =  attrib(0)
intout(1)   =  attrib(1)
intout(2)   =  attrib(2)
intout(3)   =  attrib(3)
intout(4)   =  attrib(4)
intout(5)   =  attrib(5)

ptsout(0)   =  attrib(6)
ptsout(1)   =  attrib(7)
ptsout(2)   =  attrib(8)
ptsout(3)   =  attrib(9)
```

Vqt_extend

Zum »VDI gehörende Routine für Grafikausgabe

```
$74 = 116       vqt_extent(handle, string, extent)
word                    handle;
char *                          string;
word *                                  extent;
```

Hiermit können Sie für einen gegebenen String abfragen, wie groß ein Rechteck ist, das diese Zeichenkette vollständig umschließt. Die Routine ist vor allem nützlich, um Textausgabe zu zentrieren oder an andere Grafik in der Länge anzupassen. Zeichengröße, Texteffekte und Textwinkel beeinflussen das Resultat.

Die vier Ecken des zurückgegebenen Rechtecks sind wie folgt angeordnet: Ecke 1 ist - wenn man den Text aufrecht vor sich hat - links unten, Ecke zwei ist rechts unten, Ecke 3 ist rechts oben und Ecke 4 ist links oben.

Eingabeparameter:

word handle	Handle der »virtual Workstation für Ausgabe
char *string	Zeiger auf den String, dessen Größe (und Lage) gefragt ist.
word *extent	Zeiger auf ein acht Worte umfassendes Array, in das die Rückgabeparameter eingetragen werden (s. u.).

```
contrl(0)    =  116
contrl(1)    =    0
contrl(3)    =    Länge des Strings
contrl(6)    =  handle

intin(0)     =  1. Zeichen im String
intin(1)     =  2. Zeichen im String
.
.
.
intin(n-1)   =  n. Zeichen im String
```

Ausgabeparameter:

word extent(8)	Ein acht Worte umfassendes Array, in dem nacheinander die Koordinaten der Ecken 1, 2, 3 und 4 stehen (s. o.).

```
contrl(2)    =  4
contrl(4)    =  0

ptsout(0)    =  extent(0)
ptsout(1)    =  extent(1)
.
.
.
ptsout(7)    =  extent(7)
```

Vqt_fontinfo

Zum »VDI gehörende Routine für Grafikausgabe

```
$83 = 131        vqt_fontinfo(handle, &minADE, &maxADE, distances,
word                          handle;
word *                                 minADE,  maxADE, distances;
                              &maxwidth, effects)
word *                         maxwidth, effects;
```

Auch mit dieser Routine kann man - wie mit »*vqt_width()* -Informationen über die Größe von Zeichen aus dem Zeichensatz erfragen. Allerdings wird hier der Einfluß verschiedener Texteffekte auf die Größe berücksichtigt, nicht die Größe selbst. So wird bei Kursivschrift durch die Rechtsneigung des Zeichens der umschließende Kasten um das Zeichen nach links und rechts erweitert.

Zwei Parameter bedürfen noch der genaueren Erläuterung: 'minADE' und 'maxADE'. Die Abkürzung 'ADE' steht für 'ASCII Decimal Equivalent'. Dies bedeutet jeweils den niedrigsten bzw. höchsten Wert im »ASCII-Zeichensatz. Beim Systemfont ist der niedrigste Zeichenwert Null, der höchste 255. Würde der Zeichensatz mit dem 'A' beginnen, so wäre 'minADE' gleich 65.

Begriffe wie "Base Line", "Descent Line" etc. sind bei »*vst_allignment()* näher erklärt.

Eingabeparameter:

```
word handle        Handle der »virtual Workstation für Ausgabe

word *minADE
word *maxADE
word *maxwidth     Zeiger auf Worte für Rückgabeparameter (s. u.)

word *distances
word *effects      Zeiger auf Arrays von Worten (s. u.)

contrl(0)   =  131
contrl(1)   =    0
contrl(3)   =    0
contrl(6)   =  handle
```

Ausgabeparameter:

```
word minADE        Niedrigster ASCII-Wert des Zeichens im Zei-
                   chensatz

word maxADE        Höchster Wert für ASCII-Zeichen im Zeichensatz
```

word distances(5)	Ein aus fünf Worten bestehendes Array. Die Werte haben folgende Bedeutung:
word distances(0)	Abstand der untersten Linie ("Bottom Line", diese ist identisch mit der "Top Line" des darunter liegenden Zeichens) zur Basisline ("Base Line, auf der die Buchstaben stehen - ohne Unterlängen)
word distances(1)	Abstand der Basislinie zur Unterlänge ("Descent Line")
word distances(2)	Abstand der halben Höhe ("Half distance") zur Basislinie
word distances(3)	Abstand Buchstabenobergrenze ("Ascent Line" zur Basislinie
word distances(4)	Abstand der "Top Line" zur Basislinie
word maxwidth	Maximale Breite eines das breiteste Zeichen des Zeichensatzes umschließenden Rechtecks **ohne** spezielle Texteffekte (Kursiv etc.)
word effects(3)	Ein aus drei Worten bestehendes Array mit folgender Bedeutung:
word effects(0)	Der Offset, um den der augenblickliche Texteffekt die Zeichenbreite insgesamt verbreitert
word effects(1)	Der Offset, um den der augenblickliche Texteffekt die Zeichenbreite nach links verbreitert
word effects(2)	Der Offset, um den der augenblickliche Texteffekt die Zeichenbreite nach rechts verbreitert

```
contrl(2)   =  5
contrl(4)   =  2

intout(0)   =  maxADE
intout(1)   =  minADE

ptsout(0)   =  maxwidth
ptsout(1)   =  distances(0)
ptsout(2)   =  effects(0)
ptsout(3)   =  distances(1)
ptsout(4)   =  effects(2)
ptsout(5)   =  distances(2)
ptsout(6)   =  effects(3)
ptsout(7)   =  distances(3)
ptsout(8)   =  0
ptsout(9)   =  distances(4)
```

Vqt_name

Zum »VDI gehörende Routine für Grafikausgabe

```
$82 = 130      index = vqt_name(handle, element_num, name)
word           index,           handle, element_num;
char *                                          name;
```

Diese Routine gibt einen 32-Byte langen String zurück, der den Stil des Zeichensatzes beschreibt.

Eingabeparameter:

word handle	Handle der »virtual Workstation für Ausgabe
word element_num	Nummer des Elements (eins bis zur Anzahl der verfügbaren Zeichensätze, beim Atari gibt es nur einen Zeichensatz)
char *name	Zeiger auf ein 32 Byte langen String, in den die Informationen eingetragen werden

```
contrl(0)  =  130
contrl(1)  =    0
contrl(3)  =    1
contrl(6)  =  handle

intin(0)   =  element_num
```

Ausgabeparameter:

byte name	32 Bytes an Informationen über den Stil des Zeichensatzes

```
contrl(2)   =   0
contrl(4)   =  33

intout(0)   =  Identifikationsnummer des Zeichensatzes
intout(1)   =  name(0)
.
.
.
intout(32)  =  name(31)
```

Vqt_width

Zum »VDI gehörende Routine für Grafikausgabe

```
$75 = 117       status = vtq_width(handle, character, &cell_width,
word            status,            handle;
byte                                      character;
word *                                                  cell_width;
                                   &left_delta, &right_delta)
word *                              left_delta,  right_delta;
```

Diese Routine wird benötigt, um Informationen über die Breite eines Zeichens im Zeichensatz zu erfahren. Im System-Zeichensatz ist die Breite aller Zeichen gleich, in einem eventuell nachgeladenen Zeichensatz mit Proportionalschrift jedoch nicht. Genauer gesagt erfährt man hier nicht die Breite des Zeichens selbst, sondern die Breite eines Rechtecks, das das Zeichen komplett umschließt, einschließlich eines linken Abstands zu einem links davon stehenden Zeichens ('left_delta') und einem rechten Abstand zu einem rechts davon stehenden Zeichen ('right_delta').

Eingabeparameter:

word handle	Handle der »virtual Workstation für Ausgabe
char character	Das Zeichen, über das Informationen abgefragt werden sollen.
word *cell_width word *left_delta word *right_delta	Zeiger auf Variable für Rückgabewerte (s. u.)

```
contrl(0)   =  117
contrl(1)   =    0
contrl(3)   =    1
contrl(6)   =  handle

intin(0)    =  character
```

Ausgabeparameter:

word status	Hier bekommt man entweder das Zeichen ('character') wieder zurück oder aber -1, falls das Zeichen nicht gültig war
word cell_width	Die Breite des Rechtecks, welches das Zeichen komplett umschließt
word left_delta	Abstand des Zeichens vom linken Rand des umschließenden Rechtecks

word right_delta Abstand des Zeichens vom rechten Rand des umschließenden Rechtecks. Die Breite des Zeichens ist folglich 'cell_width' minus 'left_delta' minus 'right_delta

```
contrl(2)   =  3
contrl(4)   =  1

intout(0)   =  status

ptsout(0)   =  cell_width
ptsout(1)   =  0
ptsout(2)   =  left_delta
ptsout(3)   =  0
ptsout(4)   =  right_delta
ptsout(5)   =  0
```

Vr_recfl

Zum »VDI gehörende Routine für Grafikausgabe

```
$72 = 114        vr_recfl(handle, pxyarray)
word                    handle;
word *                          pxyarray;
```

Diese Routine zeichnet ein ausgefülltes Rechteck. Im Gegensatz zu »*v_bar()* wird keine Randlinie mitgezeichnet.

Die Ausgabe wird von folgenden Routinen beeinflußt:
- »*vsf_color()*
- »*vsf_interior()*
- »*vsf_style()*
- »*vswr_mode()*

Eingabeparameter:

word handle Handle der »virtual Workstation für Ausgabe

word *pxyarray Zeiger auf ein vier Worte umfassendes Array welches die Koordinaten von zwei X-Y-Punkten enthält. Das erste Paar gibt einen beliebigen Eckpunkt des zu zeichnenden Rechtecks an, das zweite Paar den diagonal gegenüberliegenden Eckpunkt.

```
contrl(0)   =  114
contrl(1)   =    4
contrl(3)   =    2
```

```
contrl(5)   =   1
contrl(6)   =  handle

ptsin(0)    =  pxyarray[0]
ptsin(1)    =  pxyarray[1]
ptsin(2)    =  pxyarray[2]
ptsin(3)    =  pxyarray[3]
```

Ausgabeparameter: keine

```
contrl(2)   =   0
contrl(4)   =   0
```

Vr_trnfm

Zum »VDI gehörende Routine für Grafikausgabe

```
$6E = 110       vr_trnfm(handle, psrcMFDB, pdesMFDB)
word                     handle;
word *                           psrcMFDB, pdesMFDB;
```

Für »Raster gibt es zwei Formate: das Standard-Format und das geräteabhängige Format. Falls Sie nur auf dem Atari programmieren, können Sie diese Routine völlig vergessen. Sie wird nur dann gebraucht, wenn man »Raster von einem anderen Rechner auf den Atari übertragen möchte. Denn auf dem Atari ist das (monochrome) geräteabhängige Format mit dem Standard-Format identisch (siehe »Raster)!

Diese Routine wandelt das vorgefundene Format des Quellbereichs in das Format des Zielbereichs um.

Eingabeparameter:

word handle	Handle der »virtual Workstation für Ausgabe
word *psrcMFDB	Zeiger auf einen »Memory Form Definition Block des Quellrasters - erklärt bei dem entsprechenden Stichwort sowie bei »Raster
word *pdesMFDB	Zeiger auf einen »Memory Form Definition Block des Zielrasters - erklärt bei dem entsprechenden Stichwort sowie bei »Raster

```
contrl(0)   = 110
contrl(1)   =   0
contrl(3)   =   0
```

```
contrl(6)    =  handle
contrl(7-8)  =  psrcMFDB
contrl(9-10) =  pdesMFDB
```

Ausgabeparameter:

```
contrl(2)    =  0
contrl(4)    =  0
```

Vro_cpyfm

Zum »VDI gehörende Routine für Grafikausgabe

```
$6D = 109        vro_cpyfm(handle, wr_mode, pxyarray,
word                       handle, wr_mode;
word *                                   pxyarray;
                           psrcMFDB, pdesMFDB)
word *                     psrcMFDB, pdesMFDB;
```

Mit dieser Routine können Sie einen rechteckigen Bildschirmausschnitt von einem Teil des Video-Speichers zu einem anderen kopieren, vom Video-Speicher in den (Anwender-)Speicher, vom Speicher in den Video-Speicher oder vom Speicher zum Speicher.

Um diese Routine verwenden zu können, sollten Sie unbedingt zuerst das Stichwort »Raster gelesen haben!

Das Quellraster wird während des Kopiervorganges mit dem Ziel verknüpft. Die Verknüpfungsarten sind wiederum bei »Raster erläutert. Der Kopiervorgang geschieht »Pixel für »Pixel. Sollten sich die Bildausschnitte überlappen, so wird dies beim Kopieren berücksichtigt: der entsprechende Bildteil wird erst verändert **nachdem** er kopiert wurde.

Die Rasterroutinen sind sehr allgemein gehalten. Dadurch sind sie auch ziemlich langsam. Wenn Sie die Rasteroperationen beschleunigen wollen (ohne eigene Assembler-Routinen zu schreiben), dann können Sie ein paar Tricks verwenden:

- Lassen Sie die linke obere Ecke des Quell- und Zielrasters auf Wortadressen beginnen
- Vermeiden Sie Kopiermodi wie XOR, bei denen das Zielraster mit dem Quellraster logisch verknüpft werden muß
- Vermeiden Sie, daß Quelle und Ziel gegeneinander verschoben sind: wenn die linke Ecke der Quelle auf der X-Koordinate drei ist, sollte das Ziel bei der Koordinate drei, 19, 35 etc. liegen. Dies verhindert, daß jedes Wort beim Kopieren verschoben (geshifted) werden muß.

- Benutzen Sie eine andere Methode: um beispielsweise einen Bildschirmteil zu invertieren, können Sie die Routine »*v_bar()* benutzen mit einem XOR-Schreibmodus. Dies geht aber nur auf dem Bildschirm selbst.

Eingabeparameter:

word handle — Handle der »virtual Workstation für Ausgabe

word wr_mode — Definiert die Verknüpfung von Quell- und Zielraster. Eine genaue Beschreibung der möglichen 15 Modi finden Sie unter dem Stichwort »Raster.

word *pxyarray — Zeiger auf ein acht Worte umfassendes Array, in dem die Koordinaten von Quelle und Ziel stehen (Koordinaten werden im jeweils gültigen Koordinatensystem angegeben):

pxyarray(0) = X-Koordinate einer Ecke des Quellrasters
pxyarray(1) = Y-Koordinate einer Ecke des Zielrasters
pxyarray(2) = X-Koordinate einer Ecke des Quellrasters, die der ersten Ecke diagonal gegenüberliegt
pxyarray(3) = Y-Koordinate einer Ecke des Quellrasters, die der ersten Ecke diagonal gegenüberliegt
pxyarray(4) = X-Koordinate einer Ecke des Zielrasters
pxyarray(5) = Y-Koordinate einer Ecke des Zielrasters
pxyarray(6) = X-Koordinate einer Ecke des Zielrasters, die der ersten Ecke diagonal gegenüberliegt
pxyarray(7) = Y-Koordinate einer Ecke des Zielrasters, die der ersten Ecke diagonal gegenüberliegt

word *psrcMFDB — Zeiger auf einen »Memory Form Definition Block des Quellrasters - erklärt bei dem entsprechenden Stichwort sowie bei »Raster

word *pdesMFDB — Zeiger auf einen »Memory Form Definition Block des Zielrasters - erklärt bei dem entsprechenden Stichwort sowie bei »Raster

```
contrl(0)    =  109
contrl(1)    =    4
contrl(3)    =    1
contrl(6)    =  handle
contrl(7-8)  =  psrcMFDB
contrl(9-10) =  pdesMFDB

intin(0)     =  wr_mode

ptsin(0)     =  pxyarray(0)
ptsin(1)     =  pxyarray(1)
ptsin(2)     =  pxyarray(2)
ptsin(3)     =  pxyarray(3)
ptsin(4)     =  pxyarray(4)
ptsin(5)     =  pxyarray(5)
ptsin(6)     =  pxyarray(6)
ptsin(7)     =  pxyarray(7)
```

Ausgabeparameter: keine

```
contrl(2)    =  0
contrl(4)    =  0
```

Vrq_choice

Zum »VDI gehörende Routine

```
$1E = 30        status = vsm_choice(handle, ch_in, &ch_out)
word            status,             handle, ch_in;
word *                                          ch_out;
```

Diese Routine wartet, bis eine Funktionstaste gedrückt wird. Wurde keine Funktionstaste betätigt, so wird der eingegebene Wert zurückgegeben. Die Numerierung der Funktionstasten beginnt bei eins.

Vorbedingung: mit »*vsin_mode()* muß für dieses Gerät (choice) der »Request-Modus eingestellt worden sein.

Eingabeparameter:

```
word handle        Handle der »virtual Workstation für Ausgabe

word *choice       Zeiger auf Rückgabewert (s. u.)

contrl(0)    =  30
contrl(1)    =   0
contrl(3)    =   0
contrl(6)    =  handle
```

Ausgabeparameter:

word status	Status der Funktionstasten: 0 = keine Funktionstaste gedrückt 1 = Abfrage erfolgreich
word choice	Nummer der Funktionstaste (wenn 'status' = 1), sonst NULL (1 = erste Funktionstaste usw.)

```
contrl(2)   =  0
contrl(4)   =  status

intout(0)   =  choice
```

Vrq_locator

Zum »VDI gehörende Routine für Grafikausgabe

```
$1C = 28        vrq_locator(handle, x, y, &xout, &yout, &term)
word                        handle, x, y;
word *                                      xout,  yout,  term;
```

Mithilfe dieser Routine wird die Position des (Grafik-)Cursors abgefragt. Die Routine zeigt auf dem Bildschirm den »Mauszeiger an, selbst wenn er (z. B. mit »*v_hide_c()*) unsichtbar gemacht worden ist. Die Position des Cursors wird dann solange verfolgt, bis entweder eine der Maustasten oder aber <Return> gedrückt wird. Die (neuen) X-Y-Koordinaten der Maus werden an das Programm zurückgegeben. Der Mauszeiger kann mit der Tastatur oder aber mit der Maus bewegt werden.

Vorbedingung: mit *vsin_mode()* muß der Request-Modus eingestellt worden sein (für das Gerät).

Eingabeparameter:

word handle	Handle der »virtual Workstation für Ausgabe
word x	X-Position, die der Mauszeiger bei Aufruf der Routine einnehmen soll
word y	Y-Position, die der Mauszeiger bei Aufruf der Routine einnehmen soll
word *xout word *yout word *term	Zeiger auf Variablen für Rückgabewerte (s. u.)

```
contrl(0)   =  28
contrl(1)   =   1
contrl(3)   =   0 le
contrl(6)   =  hand

ptsin(0)    =  x
ptsin(1)    =  y
```

Ausgabeparameter:

word xout	Neue X-Position der Maus nach Drücken einer Ende-Taste
word ypout	Neue Y-Position der Maus nach Drücken einer Ende-Taste
word term	»ASCII-Wert der Endetaste, die für die Rückkehr der Routine zum aufrufenden Programm verantwortlich ist: $0D = 13: <Return> wurde gedrückt $20 = 32: Linke Maustaste wurde gedrückt $21 = 33: Rechte Maustaste wurde gedrückt

```
contrl(2)   =  1
contrl(4)   =  1

intout(0)   =  term

ptsout(0)   =  xout
ptsout(1)   =  yout
```

Vrq_string

Zum »VDI gehörende Routine

```
$1F = 31        vrq_string(handle, max_length, echo_mode, echo_xy,
word                       handle, max_length, echo_mode;
word *                                                     echo_xy;
                           &string)
char *                      string;
```

Diese Routine liest einen String im »Request-Modus ein. Angegeben wird die maximale Länge. Ist diese erreicht, so wird die Eingabe abgebrochen. Die Eingabe wird auch durch <Return> beendet. Es kann auch eingestellt werden, ob der eingelesene String auf dem »Bildschirm ausegeben wird.

Vorbedingung: mit *vsin_mode()* muß der Request-Modus eingestellt worden sein (für das Gerät).

Eingabeparameter:

word handle	Handle der »virtual Workstation für Ausgabe
word max_length	Maximale Länge des einzulesenden Strings. Ist der Wert negativ, so wird der Absolutbetrag verwendet.
word echo_mode	Stellt ein, ob der String bei der Eingabe auf dem Bildschirm ausgegeben wird oder nicht: 0 = Eingelesene Zeichen werden dargestellt 1 = Keine Darstellung
word *echo_xy	Zeiger auf ein zwei Worte umfassendes Array: echo_xy(0) = X-Koordinate der Ausgabeposition echo_xy(1) = Y-Koordinate der Ausgabeposition
char *string	Zeiger auf den String, in den die eingegebenen Zeichen eingetragen werden (s. u.)

```
contrl(0)   =  31
contrl(1)   =   1
contrl(3)   =   2
contrl(6)   =  handle

intin(0)    =  max_length
intin(1)    =  echo_mode

ptsin(0)    =  echo_xy(0)
ptsin(1)    =  echo_xy(1)
```

Ausgabeparameter:

```
contrl(2)   =  0
contrl(4)   =  Länge des eingelesenen Strings

intout(0)   =  Erstes Byte des eingelesenen Strings in den un-
               teren acht Bit
intout(1)   =  Zweites Byte
.
.
.
intout(n)   =  n. Byte des Strings
```

Vrq_valuator

Zum »VDI gehörende Routine

```
$1D = 29        vrq_valuator(handle, valuator_in, &valuator_out,
word                         handle, valuator_in;
word *                                               valuator_out;
                             &terminator)
word *                        terminator;
```

Diese Routine kann dazu benutzt werden, einen Wert entsprechend den gedrückten Cursor-Tasten (Pfeiltasten) zu verändern, bis die Eingabe mit <Return> beendet wird (»Request-Modus).

Der Wert des "Valuators" (= Variable, deren Wert verändert werden soll) wird folgendermaßen beeinflußt:

Gedrückte Taste	durchgeführte Wertveränderung
Cursor rauf	plus zehn
Cursor runter	minus zehn
Shift und Cursor rauf	plus eins
Shift und Cursor runter	minus eins

Mit anderen Worten: übergibt man an die Routine (mit 'valuator_in) einen Wert von 100, und wird "Cursor rauf" neunmal betätigt (danach <Return>), so bekommt man 190 zurück (in 'valuator_out'). Zusätzlich steht in 'terminator' noch, welche Taste zum Beenden benutzt wurde.

Vorbedingung: mit *vsin_mode()* muß der Request-Modus eingestellt worden sein (für das Gerät).

Eingabeparameter:

word handle	Handle der »virtual Workstation für Ausgabe
word valuator_in	Anfangswert des "Valuators"
word *valuator_out word *terminator	Zeiger auf Variablen für Rückgabeparameter (s. u.)

```
contrl(0)   =  29
contrl(1)   =   0
contrl(3)   =   1
contrl(6)   =  handle

intin(0)    =  valuator_in
```

Ausgabeparameter:

word valuator_out	Entsprechend den Tastendrücken veränderter Wert des "Valuators"
word terminator	»ASCII-Wert der Endetaste, die für die Rückkehr der Routine zum aufrufenden Programm verantwortlich ist: $0D = 13: <Return> wurde gedrückt $20 = 32: Linke Maustaste wurde gedrückt $21 = 33: Rechte Maustaste wurde gedrückt

```
contrl(2)   =  0
contrl(4)   =  2

intout(0)   =  valuator_out
intout(1)   =  terminator
```

Vrt_cpyfm

Zum »VDI gehörende Routine für Grafikausgabe

```
$79 = 121       vrt_cpyfm(handle, wr_mode, pxyarray,
word                      handle, wr_mode;
word *                                    pxyarray;
                          psrcMFDB, pdesMFDB, color_index)
word *                    psrcMFDB, pdesMFDB, color_index;
```

Mit dieser Routine können Sie einen rechteckigen Bildschirmausschnitt von einem Teil des Video-Speichers zu einem anderen kopieren, vom Video-Speicher in den (Anwender-)Speicher, vom Speicher in den Video-Speicher oder vom Speicher zum Speicher. Anders jedoch als »*vro_cpyfm()* wird von ein monochromer Bildbereich während des Kopiervorgangs nach Farbe transformiert. Damit ist es möglich, monochrome Bildausschnitte auch in den niedrigen Auflösungen zu verwenden. Allerdings ist das resultierende Bild auch wieder nur zweifarbig! Es findet keine Umsetzung von Graustufen statt.

Um diese Routine verwenden zu können, sollten Sie unbedingt zuerst das Stichwort »Raster gelesen haben!

Das Quellraster wird während des Kopiervorganges mit dem Ziel verknüpft. Die Verknüpfungsarten sind bei »*vswr_mode()* erklärt, und nicht bei »Raster. Der Kopiervorgang geschieht »Pixel für »Pixel. Sollten sich die Bildausschnitte überlappen, so wird dies beim Kopieren berücksichtigt: der entsprechende Bildteil wird erst verändert, **nachdem** er kopiert wurde.

Eingabeparameter:

word handle	Handle der »virtual Workstation für Ausgabe
word wr_mode	Definiert die Verknüpfung von Quell- und Zielraster. Hier sind nur die vier unter »*vswr_mode()* erläuterten zugelassen.
word *pxyarray(8)	Zeiger auf ein acht Worte umfassendes Array, in dem die Koordinaten von Quelle und Ziel stehen (Koordinaten werden im jeweils gültigen Koordinatensystem angegeben):

pxyarray(0) = X-Koordinate einer Ecke des Quellrasters
pxyarray(1) = Y-Koordinate einer Ecke des Zielrasters
pxyarray(2) = X-Koordinate einer Ecke des Quellrasters, die der ersten Ecke diagonal gegenüberliegt
pxyarray(3) = Y-Koordinate einer Ecke des Quellrasters, die der ersten Ecke diagonal gegenüberliegt
pxyarray(4) = X-Koordinate einer Ecke des Zielrasters
pxyarray(5) = Y-Koordinate einer Ecke des Zielrasters
pxyarray(6) = X-Koordinate einer Ecke des Zielrasters, die der ersten Ecke diagonal gegenüberliegt
pxyarray(7) = Y-Koordinate einer Ecke des Zielrasters, die der ersten Ecke diagonal gegenüberliegt

word *psrcMFDB	Zeiger auf einen »Memory Form Definition Block des Quellrasters - erklärt bei dem entsprechenden Stichwort sowie bei »Raster
word *pdesMFDB	Zeiger auf einen »Memory Form Definition Block des Zielrasters - erklärt bei dem entsprechenden Stichwort sowie bei »Raster
word *color_index	Zeiger auf ein zwei Worte umfassendes Array. In diesem ist definiert, welche Farbe die gesetzten Bits bzw. die nicht gesetzen Bits am Ziel haben sollen: color_index(0) = Farbindex, den die gesetzten Bits (bzw. »Pixel) am Ziel haben sollen

```
                    color_index(1) = Farbe der nicht gesetzten
                    Bits am Ziel

contrl(0)    =  121
contrl(1)    =    4
contrl(3)    =    3
contrl(6)    =  handle
contrl(7-8)  =  psrcMFDB
contrl(9-10) =  pdesMFDB

intin(0)     =  wr_mode
intin(1)     = color_index(0)
intin(2)     = color_index(1)

ptsin(0)     =  pxyarray(0)
ptsin(1)     =  pxyarray(1)
ptsin(2)     =  pxyarray(2)
ptsin(3)     =  pxyarray(3)
ptsin(4)     =  pxyarray(4)
ptsin(5)     =  pxyarray(5)
ptsin(6)     =  pxyarray(6)
ptsin(7)     =  pxyarray(7)
```

Ausgabeparameter: keine

```
contrl(2)    =  0
contrl(4)    =  0
```

Vs_clip

Zum »VDI gehörende Routine für Grafikausgabe

```
$81 = 129      vs_clip(handle, clip_flag, pxyarray)
word                   handle, clip_flag;
word *                                     pxyarray;
```

Mit dieser Routine ist es möglich, rechteckige Bildschirmausschnitte zu definieren, auf die die Grafikausgabe beschränkt wird (»Clipping). Alles, was bei Grafikoperationen über diesen Bereich hinausgeht, wird abgeschnitten. Damit ist es möglich, z. B. in einem Fenster zu zeichnen, ohne sich darum kümmern zu müssen, ob über den Fensterrand hinausgezeichnet wird.

Ob »Clipping aktiv ist oder nicht, kann über den Parameter 'clip_flag' eingestellt werden.

Eingabeparameter:

word handle	Handle der »virtual Workstation für Ausgabe
word clip_flag	Hiermit kann bestimmt werden, ob »Clipping ein- oder ausgeschaltet ist: 'clip_flag' = 0: kein »Clipping 'clip_flag' > 0: »Clipping eingeschaltet
word *pxyarray	Zeiger auf ein Array aus vier Worten, die den Bildschirmausschnitt definieren: pxyarray(0) = X-Koordinate einer Ecke des Bereichs pxyarray(1) = X-Koordinate einer Ecke des Bereichs pxyarray(2) = X-Koordinate einer schräg gegenüberliegenden Ecke pxyarray(3) = Y-Koordinate einer schräg gegenüberliegenden Ecke

```
contrl(0)   =  129
contrl(1)   =    2
contrl(3)   =    1
contrl(6)   =  handle

intin(0)    =  clip_flag

ptsin(0)    =  pxyarray(0)
ptsin(1)    =  pxyarray(1)
ptsin(2)    =  pxyarray(2)
ptsin(3)    =  pxyarray(3)
```

Ausgabeparameter: keine

```
contrl(2)   =  0
contrl(4)   =  0
```

Vs_color

Zum »VDI gehörende Routine für Grafikausgabe

```
$0E = 14        vs_color(handle, index, rgb_in)
word                     handle, index;
word *                                   rgb_in;
```

Mit dieser Routine können Sie die Farben einstellen. Es sind insgesamt 16 Farben einstellbar ('index'). Allerdings werden auf einem monochromen Ausgabegerät alle Farbeinstellungen als "Weiß" angesehen. Der Farbindex Null ist die Hintergrundfarbe.

Jede Farbe besteht aus einer Mischung von Rot, Grün und Blau ("RGB"). Der Anteil dieser Grundfarben wird mit Werten von 0 bis 1000 angegeben (ein Wert kleiner Null oder größer 1000 wird auf 1000 gesetzt).

Wenn das Gerät nicht über eine Farbpalette verfügt oder der Farbindex außerhalb des zulässigen Bereichs liegt, dann wird keine Aktion durchgeführt.

Mit der Routine »v*q_extnd()* kann abgefragt werden, ob überhaupt eine Farbpalette existiert; mit »*vq_color()* können die Farbwerte (Farbanteile) der Farben abgefragt werden.

Eingabeparameter:

```
word handle        Handle der »virtual Workstation für Ausgabe

word index         Farbindex

word *rgb_in       Zeiger auf ein Array aus drei Wörtern:
                   rgb_in(0) = Rotanteil
                   rgb_in(1) = Grünanteil
                   rgb_in(2) = Blauanteil

contrl(0)   =  14
contrl(1)   =   0
contrl(3)   =   4
contrl(6)   =  handle

intin(0)    =  index
intin(1)    =  rgb_in(0)
intin(2)    =  rgb_in(1)
intin(3)    =  rgb_in(2)
```

Ausgabeparameter: keine

```
contrl(2)   =  0
contrl(4)   =  0
```

Vs_curaddress

Zum »VDI gehörende Escape-Routine für Text

```
$05 = 5        v_curaddress(handle, row, columns)
word                        handle, row, column;
```

Mit dieser Routine ist es möglich, den Textcursor direkt zu positionieren. Dazu wird die Zeile und die Spalte angegeben. Befinden sich die Werte für Zeile und/oder Spalte außerhalb des zulässi-

gen Bereichs, so wird stattdessen der nächstmögliche Wert genommen.

Eingabeparameter:

word handle	Handle der »virtual Workstation für Ausgabe
word row	Zeile, in der der Textcursor positioniert werden soll.
word column	Spalte, in der der Textcursor positioniert werden soll.

```
contrl(0)   =  5
contrl(1)   =  0
contrl(3)   =  2
contrl(5)   =  11
contrl(6)   =  handle

intin(0)    =  row
intin(1)    =  column
```

Ausgabeparameter: keine

```
contrl(2)   =  0
contrl(4)   =  0
```

Vsc_form

Zum »VDI gehörende Routine für Grafikausgabe

```
$6F = 111       vsc_form(handle, pcur_form)
word                     handle;
word *                           pcur_form;
```

Diese Routine wird benutzt, um den »Mauszeiger zu verändern. Sie ist größtenteils identisch mit »*graf_mouse()*. Unter diesem Stichwort bekommen Sie nähere Informationen.

Eingabeparameter:

word handle	Handle der »virtual Workstation für Ausgabe
word *pcur_form	Zeiger auf ein 37 Byte umfassendes Array, das die Daten der Mausform enthält. Der Aufbau der Daten ist unter dem Stichwort »Mauszeiger erklärt.

```
contrl(0)   =  111
contrl(1)   =    0
contrl(3)   =   37
contrl(6)   =  handle

intin(0)    =  pcur_form(0)
.
.
.
intin(36)   =  pcur_form(36)
```

Ausgabeparameter: keine

```
contrl(2)   =  0
contrl(4)   =  0
```

Vsf_color

Zum »VDI gehörende Routine für Grafikausgabe

```
$19 = 25        set_color = vsf_color(handle, color_index)
word            set_color,            handle, color_index;
```

Hiermit ist es möglich, die Ausgabefarbe für die Füllmuster zu setzen. Bei monochromem Monitor sind dies zwei Farben (0 = weiß, 1 = schwarz), bei mittlerer Auflösung vier und bei niedriger 16 Farben. Wird ein Farbindex außerhalb des zulässigen Bereichs gewählt, so wird stattdessen der Wert eins genommen.

Eingabeparameter:

```
word handle        Handle der »virtual Workstation für Ausgabe

word color_index   Gewünschte Farbe für die Füllmuster

contrl(0)   =  25
contrl(1)   =   0
contrl(3)   =   1
contrl(6)   =  handle

intin(0)    =  color_index
```

Ausgabeparameter:

```
word set_color     Tatsächlich gesetzte Füllmusterfarbe
```

```
contrl(2)   =  0
contrl(4)   =  1

intout(0)   =  set_color
```

Vsf_interior

Zum »VDI gehörende Routine für Grafikausgabe

```
$17 = 23        set_interior = vsf_interior(handle, style)
word            set_interior,               handle, style;
```

Diese Routine legt den Stil der Füllmuster fest. In Anhang A, Abb. 1, können Sie den Zusammenhang zwischen gesetztem Stil und dem Resultat sehen.

Eingabeparameter:

```
word handle        Handle der »virtual Workstation für Ausgabe

word style         Der Stil der Füllmuster:
                   0 = Hohl                (Muster ist ein Block
                                           in der gültigen Vorder-
                                           grundfarbe)
                   1 = Solide              (Muster ist ein Block
                                           in der gültigen Hinter-
                                           grundfarbe)
                   2 = Gemustert           (siehe Abb. 1, Anhang A)
                   3 = Gestreift           (siehe Abb. 1, Anhang A)
                   4 = Benutzerdefiniert   (siehe »vsf_udpat())

contrl(0)   =  23
contrl(1)   =   0
contrl(3)   =   1
contrl(6)   =  handle

intin(0)    =  style
```

Ausgabeparameter:

```
word set_interior  Gewählter Füllstil

contrl(2)   =  0
contrl(4)   =  1

intout(0)   =  set_interior
```

Vsf_perimeter

Zum »VDI gehörende Routine für Grafikausgabe

```
$68 = 104        set_perimeter = vsf_perimeter(handle, per_vis)
word             set_perimeter,                handle, per_vis;
```

Mit dieser Routine kann eingestellt werden, ob bei z. B. bei »*v_fillarea()* der Rand als eine solide Linie gezeichnet wird oder nicht.

Eingabeparameter:

```
word handle        Handle der »virtual Workstation für Ausgabe

word per_vis       Gibt an, wie der Rand gezeichnet werden soll:
                   0  = Rand unsichtbar
                   >0 = Rand als Linie zeichnen

contrl(0)    =  104
contrl(1)    =    0
contrl(3)    =    1
contrl(6)    =  handle

intin(0)     =  per_vis
```

Ausgabeparameter:

```
word set_perimeter Gewählte Sichtbarkeit des Randes

contrl(2)    =  0
contrl(4)    =  1

intout(0)    =  set_perimeter
```

Vsf_style

Zum »VDI gehörende Routine für Grafikausgabe

```
$18 = 24         set_style = vsf_style(handle, style_index)
word             set_style,            handle, style_index;
```

Wenn mit »*vsf_interior()* der Füllstil auf gemustert oder gestreift gesetzt wurde, so kann mit dieser Routine die Art des Musters bestimmt werden. Die Abb. 1 im Anhang A sagt mehr als viele Worte. Ist der Füllstil solide, hohl oder benutzerdefiniert, so hat diese Routine keinen Einfluß. Erst wenn der Füllstil wieder auf gemustert oder gestreift gesetzt wird, kommt die gesetzte Musterart (wieder) zur Geltung.

Eingabeparameter:

word handle Handle der »virtual Workstation für Ausgabe

word style_index Siehe Abb. 1, Anhang A

```
contrl(0)   =  24
contrl(1)   =   0
contrl(3)   =   1
contrl(6)   =  handle

intin(0)    =  style_index
```

Ausgabeparameter:

word set_style Für gemusterten oder gestreiften Füllstil gültige Füllart

```
contrl(2)   =  0
contrl(4)   =  1

intout(0)   =  set_style
```

Vsf_updat

Zum »VDI gehörende Routine für Grafikausgabe

```
$70 = 112      vsf_udpat(handle, pfill_pat, planes)
word                     handle,            planes;
word *                            pfill_pat;
```

Diese Routine wird benutzt, um ein eigenes Füllmuster zu kreieren (welches z. B. von Routinen wie »*v_fillarea()*, »*v_bar()* etc. verwendet werden kann).

Mit »*vsf_interior()* kann dieses Muster eingestellt werden.

Bit 15 des ersten Wortes von 'pfill_pat ist das obere linke Bit des Musters. Bit NULL des 16. Wortes ist rechte untere Bit des Musters. Ein gesetztes Bit entspricht einem gesetzten »Pixel.

Für mehrfarbige Muster kann für jede weitere Farbebene ein weiteres Muster definiert werden (Hohe Auflösung: eine Farbebene, mittlere Auflösung: maximal zwei Farbebenen, niedrige Auflösung: maximal vier Farbebenen).

ACHTUNG: bei farbigen Mustern sollte der Schreibmodus (mit »*vswr_mode()*) auf "Überschreiben" (Modus eins) gesetzt werden,

da die sonst entstehenden Farbmuster durch unkontrollierte Farbeffekte entstellt werden.

Eingabeparameter:

word handle	Handle der »virtual Workstation für Ausgabe
long pfill_pat	Ein (mindestens) 16 Worte umfassendes Array, das das Muster enthält. Für jede Farbebene kommen weitere 16 Worte hinzu; bei niedriger Auflösung und vier Farbebenen besteht es beispielsweise aus 64 Worten
word planes	Anzahl der Farbebenen, die das Muster umfaßt

```
contrl(0)    =  112
contrl(1)    =    0
contrl(3)    =  16 * planes
contrl(6)    =  handle

intin(0)     =  pfill_pat(0)
intin(1)     =  pfill_pat(1)
.
.
.
intin(16*n-1) = Muster für n-te Farbebene
```

Ausgabeparameter: keine

```
contrl(2)    =  0
contrl(4)    =  0
```

Vsin_mode

Zum »VDI gehörende Routine für Grafikausgabe

```
$21 = 33        vsin_mode(handle, dev_type, mode)
word                      handle, dev_type, mode;
```

»VDI kennt eine Reihe von Eingabe-Modi. Mit dieser Routine wird für eines der logischen Eingabegeräte einer bestimmten Workstation (siehe »*v_opnvwk()*) der Eingabemodus bestimmt. Diese können auf ihren Status abgefragt werden (»Sample) oder auf eine Benutzereingabe warten (»Request). Folgende Logische Eingabegeräte sind vorhanden:

- locator — Positionierbares Gerät (hier: die »Maus)
- valuator — Eingabegerät, das den Wert einer Variablen ändert
- choice — Funktionstasten
- string — Tastatur zum Einlesen einer Zeichenkette (String)

Folgende Funktionen werden in ihrem Verhalten von *vsin_mode()* beeinflußt:

- »*vrq_choice()* - »*vsm_choice()*
- »*vrq_locator()* - »*vsm_locator()*
- »*vrq_string()* - »*vsm_string()*
- »*vrq_valuator()* - »*vsm_valuator()*

Jede Funktion ist doppelt vorhanden: einmal mit "vrq_" und einmal mit "vsm_" beginnend. "vrq_" steht für "Request", "vsm_" für "Sample". Die Routinen haben jeweils **identische Funktionsnummern**. Welche ausgeführt wird, hängt von vorhergehenden Aufrufen von *vsin_mode()* für die entsprechende »virtual Workstation ab.

Eingabeparameter:

word handle	Handle der »virtual Workstation für Ausgabe
word dev_type	Art des logischen Eingabegeräts: 1 = Locator 2 = Valuator 3 = Choice 4 = String
word mode	Eingabemodus: 1 = »Request 2 = »Sample

```
contrl(0)   =  33
contrl(1)   =   0
contrl(3)   =   2
contrl(6)   =  handle

intin(0)    =  dev_type
intin(1)    =  mode
```

Ausgabeparameter: keine

```
contrl(2)   =  0
contrl(4)   =  1

intout(0)   =  Gewählter Eingabemodus
```

Vsl_color

Zum »VDI gehörende Routine für Grafikausgabe

```
$11 = 17        set_color = vsl_color(handle, color_index)
word            set_color,            handle, color_index;
```

Mit dieser Routine wird die Farbe für die im folgenden zu zeichnenden Linien festgelegt. Im monochromen Modus gibt es nur die "Farben" Null und Eins (Weiß und Schwarz), bei mittleren Auflösung sind es vier und bei niedriger Auflösung 16 Farben. Ist die gewählte Farbe außerhalb des zulässigen Bereichs, so wird die Farbe auf Eins gesetzt.

Eingabeparameter:

```
word handle          Handle der »virtual Workstation für Ausgabe

word color_index     Farbe für die Linien

contrl(0)    =  17
contrl(1)    =   0
contrl(3)    =   1
contrl(6)    =  handle

intin(0)     =  color_index
```

Ausgabeparameter:

```
word set_color       Tatsächlich gesetzte Farbe (falls gewünschte
                     Farbe außerhalb des zulässigen Bereichs)

contrl(2)    =  0
contrl(4)    =  1

intout(0)    =  set_color
```

Vsl_ends

Zum »VDI gehörende Routine für Grafikausgabe

```
$6C = 108      vsl_ends(handle, beg_style, end_style)
word                    handle, beg_style, end_style
```

Die Enden einer gezeichneten Linie können verschiedene Formen haben. Normalerweise sind die Enden rechteckig, aber sie können mit dieser Routine auch auf abgerundet oder auf pfeilform gesetzt werden. Dies kann für den Linienbeginn und das Linienende getrennt eingestellt werden.

Eingabeparameter:

```
word handle          Handle der »virtual Workstation für Ausgabe
```

```
word beg_style      Aussehen des Linienanfangs:
                    0 = rechteckig
                    1 = Pfeil
                    2 = abgerundet

word end_style      Wie 'beg_style', aber für das Ende der Linie

contrl(0)   =  108
contrl(1)   =    0
contrl(3)   =    2
contrl(6)   =  handle

intin(0)    =  beg_style
intin(1)    =  end_style
```

Ausgabeparameter: keine

```
contrl(2)   =  0
contrl(4)   =  0
```

Vsl_type

Zum »VDI gehörende Routine für Grafikausgabe

```
$0F = 15        set_type = vsl_type(handle, style)
word            set_type,           handle, style;
```

Mit dieser Routine kann der Linientyp für alle weiteren Routinen gesetzt werden, die Linien zeichnen. Es gibt sechs vordefinierte Linientypen. Mit »*vsl_udsty()* kann ein eigener Linientyp bestimmt werden.

Wenn eine Linie gezeichnet wird, so wird eine Maske benutzt. Ist in der Maske ein Bit gesetzt, so wird an der Stelle die Linie in der mit »*vsl_color()* bestimmten Linienfarbe gezeichnet. Ist in der Maske ein Bit Null, so wir die Linie an der Stelle nicht gezeichnet, sondern der Hintergrund scheint durch. Zusätzlich bestimmt aber auch noch der Zeichenmodus (siehe »*vswr_mode()*), was als Vorder- und was als Hintergrund gilt. Die Maske selbst besteht aus einem Wort. Bit 15 (das am weitesten links stehende Bit) beeinflußt dabei das am weitesten links stehende »Pixel der Linie (unabhängig davon, von wo nach wo die Linie gezeichnet wird!). Nach 16 Pixeln wiederholt sich das Muster.

Folgendermaßen sind die Linientypen vordefiniert:

style	Bitmuster	Erläuterung
1	1111111111111111	durchgehende Linie
2	1111111111110000	langer Strich, kurze Unterbrechung
3	1110000011100000	punktierte Linie
4	1111111000111000	Strich - Punkt
5	1111111100000000	gestrichelte Linie
6	1111000110011000	Strich, Punkt, Punkt, Strich
7		definiert durch »*vsl_udsty()*

Eingabeparameter:

word handle Handle der »virtual Workstation für Ausgabe

word style Linientyp (s. o.)

```
contrl(0)   =  15
contrl(1)   =   0
contrl(3)   =   1
contrl(6)   =  handle

intin(0)    =  style
```

Ausgabeparameter:

word set_type Ausgewählter Linientyp

```
contrl(2)   =  0
contrl(4)   =  1

intout(0)   =  set_type
```

Vsl_udsty

Zum »VDI gehörende Routine für Grafikausgabe

```
$71 = 113       vsl_udsty(handle, pattern)
word                      handle, pattern;
```

Mit dieser Routine kann der benutzer-definierte Linientyp gesetzt werden (siehe auch »*vsl_type()*). Das Bitmuster von 'pattern' definiert dabei den Linientyp. 'pattern' = %1010101010101010 würde etwa eine gepunktete Linie erzeugen. Man beachte, daß immer sämtliche 16 Bit des Bitmusters den Linientyp beeinflussen!

Eingabeparameter:

word handle Handle der »virtual Workstation für Ausgabe

word pattern Bitmuster des benutzer-definierten Linientyps

```
contrl(0)   =  113
contrl(1)   =    0
contrl(3)   =    1
contrl(6)   =  handle

intin(0)    =  pattern
```

Ausgabeparameter:

```
contrl(2)   =  0
contrl(4)   =  0
```

Vsl_width

Zum »VDI gehörende Routine für Grafikausgabe

```
$10 = 16        set_width = vsl_width(handle, width)
word            set_width,            handle, width;
```

Mit dieser Routine kann die Linienbreite für alle Routinen gesetzt werden, die Linien zeichnen. Leider kann die Linienbreite nur eine ungerade Zahl sein. Wenn also etwa eine Linienbreite von Vier gewünscht wird, so wird die tatsächliche Breite auf die nächstkleinere ungerade Zahl gesetzt (im Beispiel wäre das Drei).

Eingabeparameter:

word handle Handle der »virtual Workstation für Ausgabe

word width Gewünschte Linienbreite

```
contrl(0)   =  16
contrl(1)   =   1
contrl(3)   =   0
contrl(6)   =  handle

ptsin(0)    =  width
ptsin(1)    =  0
```

Ausgabeparameter:

word set_width Tatsächlich gesetzte Linienbreite

```
contrl(2)   =  1
contrl(4)   =  0
```

```
ptsout(0)   =  set_width
ptsout(1)   =  0
```

Vsm_choice

Zum »VDI gehörende Routine

```
$1E = 30        status = vsm_choice(handle, &choice)
word            status,             handle;
word *                                      choice;
```

Diese Routine testet, ob bei Aufruf eine der Funktionstasten gedrückt wurde. Ist das der Fall, so wird die Nummer der Taste zurückgegeben. Die Numerierung der Funktionstasten beginnt bei eins.

Vorbedingung: mit »*vsin_mode()* muß für dieses Gerät (choice) der »Sample-Modus eingestellt worden sein.

Eingabeparameter:

```
word handle        Handle der »virtual Workstation für Ausgabe

word *choice       Zeiger auf Rückgabewert (s. u.)

contrl(0)   =  30
contrl(1)   =   0
contrl(3)   =   0
contrl(6)   =  handle
```

Ausgabeparameter:

```
word status        Status der Funktionstasten:
                   0 = keine Funktionstaste gedrückt
                   1 = Abfrage erfolgreich

word choice        Nummer der Funktionstaste (wenn 'status' = 1),
                   sonst NULL (1 = erste Funktionstaste usw.)

contrl(2)   =  0
contrl(4)   =  status

intout(0)   =  choice
```

Vsm_color

Zum »VDI gehörende Routine für Grafikausgabe

```
$14 = 20        set_color = vsm_color(handle, color_index)
word            set_color,            handle, color_index;
```

Hiermit ist es möglich, die Ausgabefarbe für Marker zu setzen. Bei monochromem Monitor sind dies zwei Farben (0 = weiß, 1 = schwarz), bei mittlerer Auflösung vier und bei niedriger 16 Farben. Wird ein Farbindex außerhalb des zulässigen Bereichs gewählt, so wird stattdessen der Wert Eins genommen.

Eingabeparameter:

```
word handle         Handle der »virtual Workstation für Ausgabe

word color_index    Gewünschte Farbe für Marker

contrl(0)   =  20
contrl(1)   =   0
contrl(3)   =   1
contrl(6)   =  handle

intin(0)    =  color_index
```

Ausgabeparameter:

```
word set_color      Tatsächlich gesetzte Markerfarbe

contrl(2)   =  0
contrl(4)   =  1

intout(0)   =  set_color
```

Vsm_height

Zum »VDI gehörende Routine für Grafikausgabe

```
$13 = 19        set_height = vsm_height(handle, height)
word            set_height,            handle, height;
```

Mit dieser Routine kann die Höhe eines Markers bestimmt werden (siehe auch »*v_pmarker()* bzw. »*vsm_type()*). Die Höhe wird in »Pixeln angegeben. Wenn die gewählte Höhe nicht darstellbar ist, so wählt »VDI die nächstkleinere mögliche Höhe.

Eingabeparameter:

```
word handle         Handle der »virtual Workstation für Ausgabe
```

word height Höhe des Markers

```
contrl(0)   =  19
contrl(1)   =   1
contrl(3)   =   0
contrl(6)   =  handle

ptsin(0)    =  0
ptsin(1)    =  height
```

Ausgabeparameter:

```
contrl(2)   =  1
contrl(4)   =  0

ptsout(0)   =  Enthält die (von »VDI) gewählte Breite des Markers
ptsout(1)   =  Enthält die (von »VDI) gewählte Höhe des Markers
```

Vsm_locator

Zum »VDI gehörende Routine

```
$1C = 28        status = vsm_locator(handle, x, y, &xout, &yout,
word            status,              handle, x, y;
word *                                                xout,  yout;
                                     &term)
word *                                term;
```

Diese Routine fragt ab (Sample-Modus!), ob die Maus bewegt wurde oder die <Return>-Taste gedrückt wurde. Der Mauscursor wird **nicht** angezeigt. Benutzen Sie »*v_show_c()*, wenn Sie dies wünschen - falls er abgeschaltet war.

Eingabeparameter:

word handle Handle der »virtual Workstation für Ausgabe

word x X-Position, die der Mauszeiger bei Aufruf der Routine einnehmen soll

word y Y-Position, die der Mauszeiger bei Aufruf der Routine einnehmen soll

word *xout
word *yout
word *term Zeiger auf Variablen für Rückgabewerte (s. u.)

```
contrl(0)   =  28
contrl(1)   =   1
contrl(3)   =   0
contrl(6)   =  handle
```

ptsin(0) = x
ptsin(1) = y

Ausgabeparameter:

word status — Diese Variable gibt an, was geschehen ist. Ihre Zusammensetzung hängt von dem Ereignis ab: status = contrl(2) ODER contrl(4) * 2 Wobei "ODER" eine **bitweise** ODER-Verknüpfung darstellt. Tabellarisch sieht das Ganze so aus:

Eingetretenes Ereignis	contrl(2)	contrl(4)	status
Mauskoordinaten geändert	1	0	1
Taste gedrückt, Maus nicht bewegt	0	1	2
Keine Eingabe	0	0	0
Taste gedrückt und Maus bewegt	1	1	3

word xout — Neue X-Position der Maus nach Drücken einer Ende-Taste

word ypout — Neue Y-Position der Maus nach Drücken einer Ende-Taste

word term — »ASCII-Wert der Endetaste, die für die Rückkehr der Routine zum aufrufenden Programm verantwortlich ist:
$0D = 13: ‹Return› wurde gedrückt
$20 = 32: Linke Maustaste wurde gedrückt
$21 = 33: Rechte Maustaste wurde gedrückt

contrl(2) = 1 - Mauskoordinaten haben sich geändert.
0 - Mauskoordinaten haben sich nicht geändert

contrl(4) = 0 - Keine Taste gedrückt
1 - Taste gedrückt

intout(0) = term

ptsout(0) = x
ptsout(1) = y

Vsm_string

Zum »VDI gehörende Routine

```
$1F = 31        vsm_string(handle, max_length, echo_mode, echo_xy,
word                       handle, max_length, echo_mode;
word *                                                     echo_xy;
                           &string)
char *                      string;
```

Diese Routine liest einen String im »Sample-Modus ein. Angegeben werden kann maximale Länge. Die Eingabe wird abgebrochen, wenn:
- keine Daten mehr vorhanden sind
- <Return> gedrückt wurde
- das intout-Array voll ist

Eingabeparameter:

word handle	Handle der »virtual Workstation für Ausgabe
word max_length	Maximale Länge des einzulesenden Strings
word echo_mode	Stellt ein, ob der String bei der Eingabe auf dem Bildschirm ausgegeben wird oder nicht: 0 = Eingelesene Zeichen werden dargestellt 1 = Keine Darstellung (Die Darstellung erfolgt mit den gültigen Text-Attributen)
word *echo_xy	Zeiger auf ein zwei Worte umfassendes Array: echo_xy(0) = X-Koordinate der Ausgabeposition echo_xy(1) = Y-Koordinate der Ausgabeposition
char *string	Zeiger auf den String, in den die eingegebenen Zeichen eingetragen werden (s. u.)

```
contrl(0)   =  31
contrl(1)   =   1
contrl(3)   =   2
contrl(6)   =  handle

intin(0)    =  max_length
intin(1)    =  echo_mode

ptsin(0)    =  echo_xy(0)
ptsin(1)    =  echo_xy(1)
```

Ausgabeparameter:

```
contrl(2)    =  0
contrl(4)    =  Länge des eingelesenen Strings
                0  = Abfrage erfolglos
                >0 = Abfrage erfolgreich (Zeichen eingelesen)

intout(0)    =  Erstes Byte des eingelesenen Strings in den un-
                teren acht Bit (falls 'contrl(4)' größer NULL)
intout(1)    =  Zweites Byte
.
.
.
intout(n)    =  n. Byte des Strings
```

Vsm_type

Zum »VDI gehörende Routine für Grafikausgabe

```
$12 = 18        set_type = vsm_type(handle, symbol)
word            set_type,           handle, symbol;
```

Mit dieser Routine kann der Typ der Marker für »*v_pmarker()* bestimmt werden. Sechs verschieden Typen können gesetzt werden. Typ Eins ist ein Punkt (damit ist plotten auf dem Bildschirm möglich). Typ Eins ist in seiner Größe unveränderlich. Die Größe der anderen kann (in Grenzen) verändert werden (siehe »*vsm_height()*).

Eingabeparameter:

```
word handle     Handle der »virtual Workstation für Ausgabe

word symbol     Typ des Markers:
                1 = .   Punkt
                2 = +   Plus
                3 = *   Stern
                4 = O   Quadrat
                5 = X   Diagonales Kreuz
                6 = <>  Diamant (auf der Spitze stehendes
                        Rechteck)
                Wenn 'symbol' außerhalb des definierten Be-
                reichs liegt, wird Typ drei (Stern) genommen)

contrl(0)    =  18
contrl(1)    =   0
contrl(3)    =   1
contrl(6)    =  handle

intin(0)     =  symbol
```

Ausgabeparameter:

word set_type Gesetzter Markertyp

```
contrl(2)   =  0
contrl(4)   =  1

intout(0)   =  set_type
```

Vsm_valuator

Zum »VDI gehörende Routine für Grafikausgabe

```
$1D = 29        vsm_valuator(handle, val_in, &val_out, &term,
word                         handle, val_in;
word *                                          val_out,  term;
                             &status)
word *                        status;
```

Diese Routine macht fast dasselbe wie »*vrq_valuator()*, nur wird *vsm_valuator()* im »Sample-Modus benutzt, d. h. die Routine wartet **nicht** bis eine Ende-Taste (<Return> oder Mausknopf) betätigt wurde, sondern kehrt gleich zurück. Ansonsten gilt das Gleiche wie bei »*vrq_valuator()*.

Eingabeparameter:

word handle Handle der »virtual Workstation für Ausgabe

word val_in Startwert des "Valuators"

word *val_out
word *term
word *status Zeiger auf Rückgabewerte (s. u.)

```
contrl(0)   =  29
contrl(1)   =   0
contrl(3)   =   1
contrl(6)   =  handle

intin(0)    =  val_in
```

Ausgabeparameter:

word val_out Neuer Wert des "Valuators" entsprechend den gedrückten Tasten **wenn überhaupt** eine betätigt wurde

```
word term          »ASCII-Wert der Endetaste, die für die Rück-
                   kehr der Routine zum aufrufenden Programm ver-
                   antwortlich ist:
                   $0D = 13: <Return> wurde gedrückt
                   $20 = 32: Linke Maustaste wurde gedrückt
                   $21 = 33: Rechte Maustaste wurde gedrückt
                   (Falls eine dieser Tasten betätigt wurde!)

word status        0 = nichts passiert
                   1 = "Valuator" verändert
                   2 = Taste wurde gedrückt

contrl(2)   =  0
contrl(4)   =  status

intout(0)   =  val_out
intout(1)   =  term
```

Vst_allignment

Zum »VDI gehörende Routine für Grafikausgabe

```
$27 = 39        vst_alignment(handle, hor_in, vert_in,
word                          handle, hor_in, vert_in;
                              &hor_out, &vert_out)
word *                         hor_out,  ver_out;
```

Hiermit kann die vertikale und horizontale Textausrichtung gewählt werden.

Erläuterung: Im Zeichensatz beanspruchen alle Zeichen denselben Platz, egal ob es sich um ein "i" oder ein "W" handelt (dies gilt für einen eventuell nachgeladenen proportionalen Zeichensatz nicht!). Das Zeichen selbst nimmt nicht den ganzen Raum ein, da links, rechts, oben und unten noch Platz für Leerraum zwischen den Zeichen sein muß (siehe »Font). D. h. das Zeichen steht in einem Rechteck. Die obere Linie des Rechtecks bezeichnet man als "Top Line", die untere als "Bottom Line". Ein Zeichen ohne Unterlänge steht auf der "Base Line", die Unterlänge selbst endet bei der "Descent Line", die sich unter der "Base Line" befindet. Das obere Ende des Zeichens berührt die "Ascent Line". Weiterhin gibt es noch die "Half Line", die das Zeichen auf halber Höhe schneidet.

Wenn nun bei der Routine »*v_gtext()* die X-Y-Koordinate der Textausgabe angeben wird, wo ist dieser Punkt in Relation zum ersten Zeichen? Solange man das mit der Routine *vst_allignment()* noch nicht geändert hat, ist der horizontale Punkt auf der "Base Line", die vertikale eine Linie (senkrecht zur "Base Line"), die

das Zeichen links berührt. Gibt man mit diesen Werten eine Zeichenkette an der Position 0,0 aus, so sieht man auf dem Bildschirm (oben) nur die Unterlängen des Strings und die »Pixel, die sich auf der "Base Line" befinden.

Mit *vst_allignment()* kann dies geändert werden. Setzt man beispielsweise die vertikale Textausrichtung auf "Top Line", so ist eine Zeichenkette, die an der Position 0,0 ausgegeben wird, vollständig auf dem Bildschirm sichtbar. Sie befindet sich an derselben Position, als hätte man sie mit einer »BIOS oder »Gemdos-Routine in die oberste Zeile geschrieben (Verwendung derselben Zeichengröße vorausgesetzt).

Eingabeparameter:

word handle — Handle der »virtual Workstation für Ausgabe

word hor_in — Gewünschte horizontale Ausrichtung:
0 = linke Ausrichtung (voreingestellt)
1 = mittig ausgerichtet
2 = rechts ausgerichtet

word vert_in — Gewünschte vertikale Ausrichtung:
0 = Base Line (voreingestellt)
1 = Half Line
2 = Ascent Line
3 = Bottom Line
4 = Descent Line
5 = Top Line
Bei Angabe einer ungültige Ausrichtung werden jeweils die mit "voreingestellt" gekennzeichneten Werte genommen!

```
contrl(0)   =  39
contrl(1)   =   0
contrl(3)   =   2
contrl(6)   =  handle

intin(0)    =  hor_in
intin(1)    =  vert_in
```

Ausgabeparameter:

word hor_out — Eingestellte horizontale Ausrichtung

word vert_out — Eingestellte vertikale Ausrichtung

```
contrl(2)   =  0
contrl(4)   =  2
```

```
intout(0)   =  hor_out
intout(1)   =  vert_out
```

Vst_color

Zum »VDI gehörende Routine für Grafikausgabe

```
$16 = 22        set_color = vst_color(handle, color_index)
word            set_color,            handle, color_index;
```

Hiermit ist es möglich, die Ausgabefarbe für Textfunktionen zu setzen. Bei monochromem Monitor sind dies zwei Farben (0 = weiß, 1 = schwarz), bei mittlerer Auflösung vier und bei niedriger 16 Farben. Wird ein Farbindex außerhalb des zulässigen Bereichs gewählt, so wird stattdessen der Wert Eins genommen.

Eingabeparameter:

```
word handle        Handle der »virtual Workstation für Ausgabe

word color_index   Gewünschte Farbe für Textausgabe

contrl(0)   =  22
contrl(1)   =   0
contrl(3)   =   1
contrl(6)   =  handle

intin(0)    =  color_index
```

Ausgabeparameter:

```
word set_color     Tatsächlich gesetzte Textfarbe

contrl(2)   =  0
contrl(4)   =  1

intout(0)   =  set_color
```

Vst_effects

Zum »VDI gehörende Routine für Grafikausgabe

```
$6A = 106       set_effects = vst_effects(handle, effect)
word            set_effects,              handle, effect;
```

Mit dieser Routine kann die Art der Textausgabe für alle weiteren Aufrufe von »*v_gtext()* und »*v_justified()* bestimmt werden. Wählbar sind fünf Texteffekte in beliebiger Kombination:

- Fettschrift
- Helle Schrift
- Kursive Schrift
- Unterstrichene Schrift
- Hohlschrift

Die sonst noch bei anderen GEM-Rechnern vorhandene Schattenschrift ist auf dem Atari leider nicht vorhanden!

Eingabeparameter:

word handle — Handle der »virtual Workstation für Ausgabe

word effects — Mit diesem Parameter kann der gewünschte Texteffekt gesetzt werden. Dies geschieht Bitweise:

Bit	Texteffekt
0	Fettschrift
1	Helle Schrift
2	Kursivschrift
3	Unterstrichene Schrift
4	Hohlschrift

Durch Setzen der entsprechenden Bits können die Texteffekte kombiniert werden. So bedeutet 'effects' = $0D = %1101, daß die Schrift im folgende ***Fett, unterstrichen und kursiv*** erscheint. Ist 'effects' = 0, so wird die Schrift normal dargestellt. Nur die unteren fünf Bits von 'effects' haben eine Bedeutung.

```
contrl(0)   =  106
contrl(1)   =    0
contrl(3)   =    1
contrl(6)   =  handle

intin(0)    =  effects
```

Ausgabeparameter:

word set_effect — Tatsächlich gesetzte Schriftart

```
contrl(2)   =  0
contrl(4)   =  1

intout(0)   =  set_effect
```

Vst_font

Zum »VDI gehörende Routine für Grafikausgabe

```
$15 = 21        set_font = vst_font(handle, font)
word            set_font,           handle, font;
```

Diese Routine legt den Zeichensatz für alle weiteren Ausgaben von Grafiktext fest (z. B. mit »*v_gtext()* oder »*v_justified()*). Zeichensatz Nummer Eins ist bereits eingebaut, weitere lassen sich mit »*vst_load_fonts()* nachladen. Mit »*vqt_name()* kann man den Namen des Zeichensatzes erfragen. Das Laden von weiteren Zeichensätzen funktioniert nur, wenn »GDOS installiert worden ist und in der Datei "»ASSIGN.SYS" die entsprechenden »Fonts eingetragen wurden. Siehe »GDOS

Eingabeparameter:

```
word handle         Handle der »virtual Workstation für Ausgabe

word font           Nummer des gewünschten Zeichensatzes

contrl(0)   =  21
contrl(1)   =   0
contrl(3)   =   1
contrl(6)   =  handle

intin(0)    =  font
```

Ausgabeparameter:

```
word set_font       Gewählter Zeichensatz

contrl(2)   =  0
contrl(4)   =  1
```

Vst_height

Zum »VDI gehörende Routine für Grafikausgabe

```
$0C = 12        vst_height(handle, height,
word                       handle, height;

                           &char_width,  &char_height
word *                      char_width,   char_height;

                           &cell_width, &cell_height)
word *                      cell_width,  cell_height;
```

Mit dieser Routine kann man - ähnlich wie bei »*vst_point()* - die Größe der Textausgabe verändern. Ein Aufruf von *vst_height()* hat

allerdings nur Einfluß auf die Ausgabe von **Grafik**-Text, etwa mit den Routinen »*v_justified()* oder »*v_gtext()*.

Anders als bei »*vst_point()* läßt sich mit dieser Routine tatsächlich die Zeichenhöhe auf eine **absolute** Größe festlegen. Wenn Sie als Zeichenhöhe ('height') z. B. 19 angeben, so beträgt danach die Höhe der Zeichen exakt 19 »Pixel. Wobei aber das Maximum der Zeichenhöhe bei 27 liegt.

Hier nun eine tabellarische Übersicht über die mit 'height' einstellbaren Zeichenhöhen und die resultierenden Werte für weitere Textausgabe:

height	char_width	char_height	cell_width	cell_height
1	1	1	1	3
2	2	2	3	4
3	4	3	4	5
4	5	4	6	6
5	6	5	7	7
6	7	6	8	8
7	8	7	9	9
8	9	8	11	10
9	10	9	12	11
10	12	10	13	13
11	13	11	15	14
12	14	12	16	16
13	7	13	8	16
14	8	14	9	17
15	8	15	9	18
16	9	16	10	19
17	9	17	19	21
18	10	18	11	22
19	10	19	12	23
20	11	20	12	24
21	12	21	13	25
22	12	22	14	26
23	12	23	14	28
24	13	24	15	29
25	13	25	15	30
26	14	27	16	32

Bei höheren Werten für 'height' tut sich nichts mehr - die eingestellten Zeichengrößen bleiben wie bei 'height' = 26. Nicht alle angegebenen Werte für 'height' sind auch sinnvoll - erst ab 'height' = 4 ergibt sich eine lesbare Schrift. Dies ist die Schriftgröße, die bei »Icons verwendet wird. Bei 'height' = 13 entsteht die Normalschrift (bei monochromem Monitor - in Farbe entsteht bei 'height' = 6 Normalschrift!).

Eingabeparameter:

word handle	Handle der »virtual Workstation für Ausgabe
word height	Gewünschte Höhe der Zeichen bei weiterer Textausgabe
word *char_width word *char_height word *cell_width word *cell_height	Adressen der Variablen für Rückgabewerte (s. u.)

```
contrl(0)   =  12
contrl(1)   =   1
contrl(3)   =   0
contrl(6)   =  handle

ptsin(0)    =  0
ptsin(1)    =  height
```

Ausgabeparameter:

word char_width	Die Breite des breitesten Zeichens im »Font (Zeichensatz) in »Pixeln.
word char_height	Die Höhe des höchsten Zeichens in »Pixeln
word cell_width	Die Breite eines Rechtecks, mit dem das größte Zeichen **komplett** umschlossen werden kann (einschließlich Leerraum um das Zeichen selbst).
word cell_height	Die Höhe eines Rechtecks, mit dem das größte Zeichen **komplett** umschlossen werden kann (einschließlich Leerraum um das Zeichen selbst).

```
contrl(2)   =  2
contrl(4)   =  0

ptsout(0)   =  char_width
ptsout(1)   =  char_height
ptsout(2)   =  cell_width
ptsout(3)   =  cell_height
```

V_write_meta

Zum »VDI gehörende Escape-Routine für Metafiles

```
$05 = 5          v_write_meta(handle, num_intin, intin, num_ptsin,
word                          handle, num_intin,        num_ptsin;
word *                                                  intin;
                              ptsin)
word *                        ptsin;
```

Mit dieser Routine ist es möglich, **eigene** Metafile-Items in die Metafile-Datei zu schreiben (siehe »Metafile). Diese benutzerdefinierten Funktionen können Möglichkeiten der »Metafiles erhöhen.

Eingabeparameter:

word handle	Handle der »virtual Workstation für Ausgabe
word num_intin	Anzahl der Werte im **intin**-Array
word num_ptsin	Anzahl der Werte im **ptsin**-Array
word *intin	Zeiger auf ein Array, in dem Benutzer-definierte Werte stehen
word *ptsin	Zeiger auf ein Array, in dem Benutzer-definierte Werte stehen

```
contrl(0)   =  5
contrl(1)   =  Anzahl der Eingabe-Punkte
contrl(3)   =  num_intin
contrl(5)   =  99
contrl(6)   =  handle

intin(0)    =  Benutzer-eigener Opcode (101 oder größer)
intin(1)    =  Benutzer-definierte Information
.
.
.
intin(n)    =  Benutzer-definierte Information

ptsin(0)    =  Benutzer-definierte Information
.
.
.
ptsin(n)    =  Benutzer-definierte Information
```

Ausgabeparameter: keine

contrl(2) = 0
contrl(4) = 0

Vst_load_font

Zum »VDI gehörende Routine für Grafikausgabe

$77 = 119 **additional = vst_load_fonts(handle, select)**
word additional, handel, select;

Mit dieser Funktion können weitere »Fonts (Zeichensätze) nachgeladen werden. Die Bedingung dafür ist, daß »GDOS installiert wurde (beim Systemstart) und das in der Datei "»ASSIGN.SYS" die Namen der Zeichensätze eingetragen worden sind. Sehen Sie dazu bitte die Stichwörter »GDOS und »FONT.

Es werden **alle** in "»ASSIGN.SYS" eingetragenen Zeichensätze geladen. Mit »*vst_font()* wird der entsprechende Zeichensatz aktiviert. **Jede** der »virtual Workstation, die den Zeichensatz benutzen möchte, **muß** *vst_load_fonts()* aufrufen (mit dem entsprechenden »Handle als Parameter). Tatsächlich in den Speicher geladen werden die Zeichensätze aber nur beim ersten Aufruf der Funktion.

WICHTIG! Tragen Sie in »ASSIGN.SYS die gewünschten Zeichensätze unter der Bildschirmnummer Eins ein ("Default Screen"), falls Sie mit dem monochromem Monitor arbeiten.

Mit »*vst_unload_fonts()* können die Zeichensätze wieder entfernt werden.

Eingabeparameter:

word handle Handle der »virtual Workstation für Ausgabe

word select Reserviert für zukünftige Erweiterungen, muß auf NULL gesetzt sein

contrl(0) = 119
contrl(1) = 0
contrl(3) = 1
contrl(6) = handle

intin(0) = select

Ausgabeparameter:

word additional — Die Anzahl der neu erzeugten Zeichensatz-Nummern, die zur Identifikation benötigt werden. Wenn für die angegebene »virtual Workstation die »Fonts bereits zur Verfügung stehen, so wird hier eine NULL zurückgegeben. Die Zeichensätze werden dann nicht noch einmal geladen.

contrl(2) = 0
contrl(4) = 1

Vst_point

Zum »VDI gehörende Routine für Grafikausgabe

```
$6B = 107      set_point = vst_point(handle, point,
word                                 handle, point;

                                     &char_width,  &char_height
word *                                char_width,   char_height;

                                     &cell_width, &cell_height)
word *                                cell_width,  cell_height;
```

Mit dieser Routine kann man - ähnlich wie bei »*vst_height()* - die Größe der Textausgabe verändern. Ein Aufruf von *vst_point()* hat allerdings nur Einfluß auf die Ausgabe von **Grafik**-Text, etwa mit den Routinen »*v_justified()* oder »*v_gtext()*.

Der Parameter 'point' gibt die Höhe des Zeichens von der Basislinie des Zeichens bis zur Basislinie eines anderen Zeichens darüber (in 1/72 Zoll). Dies ist der Wert von 'cell_height' (siehe auch »*vst_point()* und »font).

Hier nun eine tabellarische Übersicht über die mit 'point' einstellbaren Zeichenhöhen und die resultierenden Werte für weitere Textausgabe:

point	char_width	char_height	cell_width	cell_height
1	5	4	6	6
9	7	6	8	8
10	7	13	8	16
16	10	9	12	12
18	14	13	16	16
20	14	27	16	16

Die Bedeutung der Parameter 'char_width' etc. wird unter "Ausgabeparameter" weiter unten erläutert.

Nur die angegeben Werte sind sinnvoll, alle anderen führen dazu, daß der nächstkleinere Wert aus der Tabelle für 'point' genommen wird. Für Zwischengrößen siehe »*vst_height()*.

Eingabeparameter:

word handle	Handle der »virtual Workstation für Ausgabe
word point	Gewünschte Höhe der Zeichen bei weiterer Textausgabe in 1/72 Zoll (=inch).
word *char_width word *char_height word *cell_width word *cell_height	Adressen der Variablen für Rückgabewerte (s. u.)

```
contrl(0)   =  107
contrl(1)   =    1
contrl(3)   =    0
contrl(6)   =  handle

intin(0)    =  point
```

Ausgabeparameter:

word set_point	Tatsächlich gewählte Zeichenhöhe
word char_width	Die Breite des breitesten Zeichens im »Font (Zeichensatz) in »Pixeln.
word char_height	Die Höhe des höchsten Zeichens in »Pixeln
word cell_width	Die Breite eines Rechtecks, mit dem das größte Zeichen **komplett** umschlossen werden kann (einschließlich Leerraum um das Zeichen selbst).
word cell_height	Die Höhe eines Rechtecks, mit dem das größte Zeichen **komplett** umschlossen werden kann (einschließlich Leerraum um das Zeichen selbst).

```
contrl(2)   =  2
contrl(4)   =  1

intout(0)   =  set_point
```

```
ptsout(0)   =  char_width
ptsout(1)   =  char_height
ptsout(2)   =  cell_width
ptsout(3)   =  cell_height
```

Vst_rotation

Zum »VDI gehörende Routine für Grafikausgabe

```
$0D = 13        set_baseline = vst_rotation(handle, angle)
word            set_baseline,                handle, angle;
```

Mit dieser Routine kann die Richtung der Textausgabe bestimmt werden. Auf dem Atari kann der Textwinkel leider nur in 90-Grad-Schritten gesetzt werden. Wie bei Winkelangaben für »VDI üblich, wird der Winkel in Zehntelgraden angegeben (siehe auch »Radius).

Diese Routine beeinflußt die Ausgabe der Routinen »*v_gtext()* und »*v_justified()*.

Eingabeparameter:

word handle	Handle der »virtual Workstation für Ausgabe
word angle	Der gewünschte Rotationswinkel in 1/10 Grad. Mögliche Werte: 0, 900, 1800 und 2700. Andere Werte führen dazu, das die bestmögliche Anpassung an den gewünschten Wert stattfindet.

```
contrl(0)   =  13
contrl(1)   =   0
contrl(3)   =   1
contrl(6)   =  handle

intin(0)    =  angle
```

Ausgabeparameter:

word set_baseline	Tatsächlich gewählter Winkel

```
contrl(2)   =  0
contrl(4)   =  1

intout(0)   =  set_baseline
```

Vst_unload_fonts

Zum »VDI gehörende Routine für Grafikausgabe

```
$78 = 120        vst_unload_fonts(handle, select)
word                              handle, select;
```

Die zusätzlichen Zeichensätze, die mit »*vst_load_font()* geladen wurden, können mit dieser Funktion wieder aus dem Speicher entfernt werden. Dies geschieht allerdings nur dann, wenn alle »virtual Workstations, die Zugriff auf die Zeichensätze haben, diese wieder "ausgeladen" haben oder geschlossen wurden.

Eingabeparameter:

word handle Handle der »virtual Workstation für Ausgabe

word select Reserviert für zukünftige Erweiterungen, muß auf NULL gesetzt sein

```
contrl(0)   =  120
contrl(1)   =    0
contrl(3)   =    1
contrl(6)   =  handle

intin(0)    =  select
```

Ausgabeparameter: keine

```
contrl(2)   =  0
contrl(4)   =  0
```

Vswr_mode

Zum »VDI gehörende Routine für Grafikausgabe

```
$20 = 32         set_mode = vswr_mode(handle, mode)
word             set_mode,            handle, mode;
```

Mit dieser Routine wird der Schreibmodus für alle folgenden Grafik-Operationen gewählt. Der Schreibmodus bestimmt, wie bereits gesetzte »Pixel auf dem »Bildschirm von der Zeichenoperation beeinflußt werden. So kann die Grafikausgabe die vorhandenen »Pixel auf dem Schirm (das Ziel der Grafikoperation) einfach überschreiben ("Replace") oder mit den vorhandenen Pixeln nach einer bestimmten Methode verknüpft werden.

Es gibt vier verschiedene Schreibmodi:

Replace: mode = 1 (Überschreiben)
Alle »Pixel werden durch die Operation überschrieben. Dieser Modus wird auch dann gewählt, wenn 'mode' außerhalb des zulässigen Bereichs liegt.

Transparent: mode = 2 (Mischen)
Bei diesem Modus scheint der Untergrund, auf dem gezeichnet wird, durch. Ziel (Bildschirm) und Quelle (Grafikoperation) werden durch eine bitweise ODER-Verknüpfung miteinander gemischt.

XOR: mode = 3 (Exklusiv-Oder)
Bei diesem Modus werden Quelle und Ziel durch eine Exklusiv-Oder-Verknüpfung miteinander gemischt. Beim monochromen Monitor führt eine zweimalige Grafikoperation derselben Art zu einer Restaurierung des vorherigen Zustands

Reverse Transparent: mode = 4 (Mischen umgekehrt)
Dieser Modus ist wie "Transparent" auch, aber die Quelle wird vor der Operation invertiert

Eingabeparameter:

word handle Handle der »virtual Workstation für Ausgabe

```
contrl(0)   =  32
contrl(1)   =   0
contrl(3)   =   1
contrl(6)   =  handle

intin(0)    =  mode
```

Ausgabeparameter:

word set_mode Tatsächlich gewählter Schreibmodus (für den Fall, daß 'mode' außerhalb des zulässigen Bereichs liegt, kann dies ein anderer Wert als 'mode' sein)

```
contrl(2)   =  0
contrl(4)   =  1

intout(0)   =  set_mode
```

VT-52-Codes

Siehe »Bildschirmsteuerung

Warnbox

siehe »Alarmbox

Wildcards

Sie können mit W. bei Diskettennamen Abkürzungen vornehmen.
Beispiel:
Sie haben folgendes »Directory auf Diskette:

B100.DAT
B200.DAT
B666.DAT
TEST.PRG
START.TOS

Wenn sie jetzt im Directory suchen wollen (siehe dazu unter den »Gemdos-Funktion *Fsfirst()* und *Fsnext()* nach) erhalten Sie:

Eingabe	Ergebnis
.	B100.DAT, B200.DAT, B666.DAT, TEST.PRG, START.TOS
*.DAT	B100.DAT, B200.DAT, B666.DAT
B?00.DAT	B100.DAT, B200.DAT

Wind_calc

Zur »Window Library gehörende Routine:

```
$6C = 108       wi_creturn = wind_calc(wi_ctype, wi_ckind,
word            wi_creturn,            wi_ctype, wi_ckind:
                                       wi_cinx, wi_ciny,
word                                   wi_cinx, wi_ciny,
                                       wi_cinw, wi_cinh,
word                                   wi_cinw, wi_cinh;
                                       &wi_coutx, &wi_couty
word *                                  wi_coutx,  wi_couty;
                                       &wi_coutw, &wi_couth)
word *                                  wi_coutw,  wi_couth;
```

Mithilfe dieser Routine können Sie berechnen, wie groß die Gesamtfläche eines »Fensters inklusive aller äußeren Bestandteile ist bei gegebener Arbeitsfläche des »Fensters ist. Sie können auch die Größe der Arbeitsfläche eines »Fensters berechnen, wenn Sie die Größe der Gesamtfläche kennen.

Eingabeparameter:

word wi_ctype	0 = Die Gesamtfläche des Fensters soll berechnet werden, die Eingabewerte geben die Größe (und Lage) der Arbeitsfläche (Fenster->Innenfläche) an. 1 = Die Größe (und Lage) der Arbeitsfläche soll berechnet werden, die Eingabewerte geben die Gesamtfläche des Fensters an inklusive aller äußeren Bestandteile
word wi_ckind	Mit diesem Parameter wird angegeben, über welche Bestandteile das Fenster überhaupt verfügt. Die Berechnung dieses Werts wird genau wie bei »*wind_create()* vorgenommen (siehe dort). wi_ckind = NAME | INFO; heiß z. B., daß das Fenster über eine »Titel- und eine »Infozeile verfügt ("|" ist die bitweise ODER-Verknüpfung zweier Werte, die Bedeutung der symbolischen Namen "NAME" und "INFO" entnehmen Sie bitte der Tabelle bei »*wi_create()*.
word wi_cinx	X-Koordinate der Arbeits/Gesamtfläche
word wi_ciny	Y-Koordinate der Arbeits/Gesamtfläche
word wi_cinw	Breite der Arbeits/Gesamtfläche
word wi_cinh	Höhe der Arbeits/Gesamtfläche
word *wi_coutx word *wi_couty word *wi_coutw word *wi_couth	Zeiger auf die Variablen für die Rückgabewerte (s. u.)

```
contrl(0)   =  108
contrl(1)   =    6
contrl(3)   =    0

int_in(0)   =  wi_ctype
int_in(1)   =  wi_ckind
int_in(2)   =  wi_cinx
int_in(3)   =  wi_ciny
int_in(4)   =  wi_cinw
int_in(5)   =  wi_cinh
```

Ausgabeparameter:

wi_creturn	0 = Ein Fehler ist passiert, n = alles Ok.
word wi_coutx	X-Koordinate der Arbeitsfläche (wi_ctype = 1) oder der Gesamtfläche (wi_ctype = 0).
word wi_couty	Y-Koordinate der Arbeitsfläche (wi_ctype = 1) oder der Gesamtfläche (wi_ctype = 0).
word wi_coutw	Breite der Arbeitsfläche (wi_ctype = 1) oder der Gesamtfläche (wi_ctype = 0).
word wi_couth	Höhe der Arbeitsfläche (wi_ctype = 1) oder der Gesamtfläche (wi_ctype = 0).

```
contrl(2)   =   5
contrl(4)   =   0

int_out(0)  =  wi_creturn
int_out(1)  =  wi_coutx
int_out(2)  =  wi_couty
int_out(3)  =  wi_coutw
int_out(4)  =  wi_couth
```

Wind_close

Zur »Window Library gehörende Routine:

```
$66 = 102       wi_clreturn = wind_close(wi_clhandle)
word            wi_clretrun,             wi_clhandle;
```

Mit dieser Routine kann ein Fenster zum Verschwinden gebracht werden - vom Bildschirm, die bei »*wind_create()* angegebenen Daten bleiben aber erhalten. Das Fenster kann mit einem erneuten »*wind_open()*-Aufruf wieder geöffnet werden. Erst mit »*wind_delete()* wird ein Fenster endgültig gelöscht (und damit die »Fensternummer für neue Fenster wieder freigegeben).

Eingabeparameter:

word wi_clhandle	Die »Fensternummer

```
contrl(0)   =  102
contrl(1)   =    1
contrl(3)   =    0

int_in(0)   =  wi_clhandle
```

Ausgabeparameter:

word wi_clreturn 0 = Ein Fehler ist passiert (z. B. ein nicht vorhandenes Fenster geschlossen), n = alles Ok

```
contrl(2)   =   1
contrl(4)   =   0

int_out(0)  =  wi_clreturn
```

Wind_create

Zur »Window Library gehörende Routine:

```
$64 = 100       wi_crreturn = wind_create(wi_crkind,
word            wi_crreturn,              wi_crkind;
                                          wi_crwx, wi_crwy,
word                                      wi_crwx, wi_crwy;
                                          wi_crww, wi_crwh)
word                                      wi_crww, wi_crwh;
```

Diese Routine weist »GEM an, ein Fenster zu erzeugen. Dabei wird im Speicher Platz geschaffen für die notwendigen Fensterdaten. Die Funktion gibt einen Wert zurück, den sog. "handle" (wörtlich: Griff), der für alle Fenster-Routinen (außer »*wind_calc()*) zur Identifikation des Fensters benötigt wird ("Window-handle", von uns als »Fensternummer bezeichnet). Der Wert ist eine positive, ganze Zahl, kein Zeiger auf irgendetwas.

»GEM ist in der Lage, insgesamt acht Fenster zu erzeugen und darzustellen (der »Desktop verwendet nur vier, um für »Desk-Accessories noch Fenster übrig zu lassen). Beim Versuch, mehr Fenster zu öffnen, gibt »GEM mit dieser Routine eine negative Zahl zurück (was man immer testen sollte, damit man den Benutzer auffordern kann, im Bedarfsfall ein vorhandenes Fenster zu schließen). Außerdem muß darauf geachtet werden, mit Beendigung des Programms alle geöffneten Fenster mittels »*wind_delete()* wieder zu löschen, da die entsprechenden »Fensternummern sonst nicht mehr zur Verfügung stehen (für niemanden!). Ein Programm ist also in der Lage, »Fensternummern zu "vernichten".

Der Desktop-Hintergrund ist ein spezielles neuntes Fenster mit der »Fensternummer NULL. Es ist immer vorhanden und kann nicht gelöscht werden, gleichzeitig begrenzt es die maximale Größe aller möglichen Fenster. Es kann ansonsten benutzt werden wie jedes andere Fenster auch.

Die Eigenschaften des Fensters (damit auch die Bestandteile) wer-

den ebenfalls mit diesem Aufruf festgelegt (siehe auch die zugehörige Abbildung im Anhang A) Folgende Möglichkeiten können (nicht ganz beliebig) kombiniert werden:

Bez.	Bit	Bedeutung
NAME	0	Ein einzeiliger Titel oben im Fensterbalken
CLOSER	1	Schließfeld links oben
FULLER	2	Vergrößerungsbox rechts oben
MOVER	3	Fenster kann bewegt werden
INFO	4	Eine zweite Zeile unter der Titelzeile
SIZER	5	Vergrößerungs/Verkleinerungsbox rechts unten
UPARROW	6	Pfeil über dem rechten Rollbalken
DNARROW	7	Pfeil unter dem rechten Rollbalken
VSLIDE	8	Vertikaler Rollbalken
LFARROW	9	Pfeil links vom horizontalen Rollbalken
RTARROW	10	Pfeil rechts vom horizontalen Rollbalken
HSLIDE	11	Horizontaler Rollbalken

Die symbolischen Namen wie "NAME", "CLOSER" etc. sind in "»GEMDEFS.H" definiert. Ist im ersten Parameter der Routine ('wi_crkind') das entsprechende Bit gesetzt, so wird von »GEM ein Fenster mit der betreffenden Eigenschaft erzeugt (aber erst mit dem Aufruf von »*wind_open()* auf dem Bildschirm dargestellt). Wobei folgende Einschränkungen zu beachten sind:

Soll das Fenster bewegt werden können ("MOVER"-Bit gesetzt), so muß es auch eine Titelzeile haben. Wird einer der Pfeile angegeben (UPARROW, DNARROW, LFARROW, RTARROW), so muß auch der zugehörige (vertikale bzw. horizontale) »Rollbalken angegeben werden, d. h., zu jeder Eigenschaft gehört ein bestimmter Fenster-Bestandteil.

Wenn Sie sich die "Nachrichten zur Fensterbehandlung" (»Message) nochmal genauer ansehen, so wird Ihnen eine Korrespondenz zwischen den Nachrichten und den eben erwähnten Fenster-Eigenschaften aufgefallen sein. Wie dort ebenfalls gesagt, kann Ihr Programm nur dann eine Nachricht für ein bestimmtes Fenster bekommen, wenn das Fenster auch über die entsprechende Eigenschaft verfügt! Wenn also ein Fenster die Eigenschaft "MOVER" nicht besitzt, so wird »GEM für das Fenster niemals die Nachricht "WM_MOVED" (= Fenster wurde bewegt) abgeben - denn der Benutzer kann die betreffende Aktion nicht durchführen.

Im allgemeinen werden die einzelnen Bestandteile (und damit Eigenschaften) des Fensters in einer bestimmten Weise interpretiert: ein Anklicken der »Vergrößerungsbox beispielsweise bedeutet, daß der Benutzer das Fenster auf volle Größe geöffnet haben möchte, oder, wenn dies bereits der Fall ist, daß das Fenster auf seine vorige Größe zurückschrumpfen soll.

»GEM führt die erwähnte Aktion aber nicht selbst durch, sondern schickt Ihrem Programm lediglich eine Nachricht, daß die »Vergrößerungsbox angeklickt worden ist. Dies gibt dem Programm die Möglichkeit, die Anforderung des Benutzers zu akzeptieren, zu verändern oder aber zurückzuweisen. So kann z. B. erreicht werden, daß das Fenster nur auf ganz bestimmte Positionen des Bildschirms verschoben werden kann, daß die Größe des Fensters stets nur ein Vielfaches von 16 ist etc.

ACHTUNG! Sollten Sie als Bestandteil 'NAME' und/oder 'INFO' angegeben haben, so **müssen** Sie **vor** dem »*wind_open()*-Aufruf erst mit »*wind_set()* die Adresse des Namens- bzw. Info-Strings an »AES übergeben! Sonst hagelt's Bomben.

Eingabeparameter:

word wi_crkind — Hiermit werden die individuellen Komponenten des Fensters festgelegt (s. o.). Die zwölf Bestandteile (und damit Eigenschaften) sind oben ausführlich erklärt.

word wi_crwx — X-Koordinate der äußeren Ecke links oben des Fensters bei voller Größe (die tatsächliche Größe der Darstellung wird erst bei »*wind_open()* festgelegt, muß aber kleiner oder gleich der hier angegebenen sein).

word wi_crwy — Y-Koordinate des Fensters

word wi_crww — Breite des Fensters in Pixeln

word wi_crwh — Höhe des Fensters in Pixeln

```
contrl(0)   =  100
contrl(1)   =    5
contrl(3)   =    0

int_in(0)   =  wi_crwkind
int_in(1)   =  wi_crwx
int_in(2)   =  wi_crwy
int_in(3)   =  wi_crww
int_in(4)   =  wi_crwh
```

Ausgabeparameter:

word wi_crreturn — Wenn alles gut ging, eine positive Zahl von 1 bis n, das sog. "»Window-handle" (von uns als "»Fensternummer" bezeichnet) mit dieser Zahl wird bei allen anderen Fenster-Routinen

dieses Fenster identifiziert. Falls »GEM kein Fenster mehr öffnen kann, wird eine negative Zahl zurückgegeben

```
contrl(2)   =   1
contrl(4)   =   0

int_out(0)  =  wi_crreturn
```

Wind_delete

Zur »Window Library gehörende Routine:

```
$67 = 103       wi_dreturn = wind_delete(wi_dhandle)
word            wi_dreturn,              wi_dhandle;
```

Mit dieser Routine kann ein »Fenster endgültig gelöscht werden. Damit wird auch der von »GEM intern für dieses Fenster belegte Speicherplatz wieder freigegeben. Übrigens sollte ein Fenster erst mit »*wind_close()* geschlossen werden, bevor es mit »*wind_delete()* in die ewigen Speichergründe geschickt wird, sonst bleibt auf dem Bildschirm das Fenster als häßliches, totes Bild zurück...

Eingabeparameter:

```
word wi_dhandle        »Fensternummer des zu löschenden »Fensters

contrl(0)   =  103
contrl(1)   =    1
contrl(3)   =    0

int_in(0)   =  wi_dhandle
```

Ausgabeparameter:

```
word wi_dreturn    »Fehlernummer: 0 = ein Fehler ist vorgekommen,
                   n = alles Ok.

contrl(2)   =    1
contrl(4)   =    0

int_out(0)  =  wi_dreturn
```

Wind_find

Zur »Window Library gehörende Routine:

```
$6A = 106        wi_freturn = wind_find(wi_fmx, wi_fmy)
word             wi_freturn,            wi_fmx, wi_fmy;
```

Mit dieser Routine können Sie herausfinden, welches Fenster sich an der Position 'wi_fmx', 'wi_fmy' befindet. Für gewöhnlich wird das dazu verwendet, herauszufinden, welches Fenster sich unter dem Mauszeiger befindet. Dazu wird mit »*graf_mkstate()* die Position (X- Y-Koordinaten) der Maus bestimmt, und die Koordinaten werden an *wind_find()* übergeben.

Eingabeparameter:

```
word wi_fmx     X-Koordinate des Mauszeigers

word wi_fmy     Y-Koordinate des Mauszeigers

contrl(0)   =  106
contrl(1)   =    2
contrl(3)   =    0

int_in(0)   =  wi_fmx
int_in(1)   =  wi_fmy
```

Ausgabeparameter:

```
word wi_freturn     Entweder die »Fensternummer unter den angege-
                    benen X- Y-Koordinaten, oder aber -1, wenn da
                    kein Fenster ist.

contrl(2)   =    1
contrl(4)   =    0

int_out(0)  =  wi_freturn
```

Wind_get

Zur »Window Library gehörende Routine:

```
$68 = 104        wi_greturn = wind_get(wi_ghandle, wi_gfield,
word             wi_greturn,           wi_ghandle, wi_gfield;
                                       &wi_gw1, &wi_gw2,
word *                                  wi_gw1,  wi_gw2;
                                       &wi_gw3, &wi_gw4)
word *                                  wi_gw3,  wi_gw4;
```

Dies ist eine Vielzweck-Routine, mit der Informationen über vor-

handene »Fenster abgefragt werden können, z. B. die Größe eines »Fensters (außen oder innen), welches »Fenster aktiv ist, Größe und Lage der »Rollbalken usw.

Im Einzelnen richtet sich das, was man bekommt, nach dem Parameter 'wi_gfield' (= Routinennummer). Daher erklären wir ausnahmsweise einmal an dieser Stelle die Eingabe - bzw. Ausgabeparameter. Die verwendeten symbolischen Namen sind in "»GEMDEFS.H" definiert. Hier die möglichen Werte von 'wi_gfield' (in der Reihenfolge Routinennummer - symbolischer Name: Erläuterung, darunter die Bedeutung der Rückgabeparameter):

4 - WF_WORKXYWH Gibt vom angegeben Fenster die Arbeitsfläche (= innere Fläche) zurück:
wi_gw1 X-Koordinate
wi_gw2 Y-Koordinate
wi_gw3 Breite
wi_gw4 Höhe

5 - WF_CURRXYWH Die gesamte Fläche des »Fensters inklusive Rahmen, »Titelbalken, »Info-Zeile etc. wird zurückgegeben:
wi_gw1 X-Koordinate
wi_gw2 Y-Koordinate
wi_gw3 Breite
wi_gw4 Höhe

6 - WF_PREVXYWH Gibt die Gesamtgröße des vorherigen Fensters inklusive Rahmen etc. zurück:
wi_gw1 X-Koordinate
wi_gw2 Y-Koordinate
wi_gw3 Breite
wi_gw4 Höhe

7 - WF_FULLXYWH Hiermit kann man die Fläche des Fensters in seiner maximalen Größe abfragen, inklusive aller Bestandteile (»Titelbalken etc.):
wi_gw1 X-Koordinate
wi_gw2 Y-Koordinate
wi_gw3 Breite
wi_gw4 Höhe

8 - WF_HSLIDE Gibt die momentane Position des horizontalen Schiebers zurück, eine Zahl von 1 bis 1000 (1 = Schieber ist ganz links, 1000 = Schieber ist ganz rechts)
wi_gw1 Position des horizontalen Schiebers

9 - WF_VSLIDE Gibt ebenfalls eine Zahl von 1 bis 1000 zurück, die Position des vertikalen Schiebers (1 = oben, 1000 = unten):
wi_gw1 Position des vertikalen Schiebers

10 - WF_TOP Erfragt die »Fensternummer des gerade aktiven (= zuoberst befindlichen) »Fensters:
wi_gw1 gefragte »Fensternummer

11 - WF_FIRSTXYWH Die Koordinaten des ersten Rechtecks der »Rechteckliste (was das im Einzelnen bedeutet, lesen Sie bitte im Abschnitt über das Neuzeichnen von »Fenstern nach):
wi_gw1 X-Koordinate
wi_gw2 Y-Koordinate
wi_gw3 Breite
wi_gw4 Höhe

12 - WF_NEXTXYWH Die Koordinaten des nächsten Rechtecks der »Rechteckliste:
wi_gw1 X-Koordinate
wi_gw2 Y-Koordinate
wi_gw3 Breite
wi_gw4 Höhe

13 - WF_RESVD keine Bedeutung - Reserviert

15 - WF_HSLSIZE Die Größe des horizontalen Schiebers relativ zur Größe des »Rollbalkens. Ein Wert von -1 bedeutet dabei, daß der Schieber eine minimale Größe hat (also quadratisch ist), sonst gehen die Werte von 1 bis 1000 (1000 = maximale Größe).
wi_gw1 Schiebergröße

16 - WF_VSLSIZE Hier gelten die gleichen Angaben wie für WF_HSLSIZE, nur handelt es sich hier um den vertikalen Schieber:
wi_gw1 Schiebergröße

17 - WF_SCREEN Hiermit können Sie herausfinden, wo sich der Puffer befindet, den »AES verwendet, um bei »Warnboxen und und »Drop-Down-Menüs den Hintergrund zwischenzuspeichern (und in den ein Viertel des Bildschirms hineinpaßt - was gleichzeitig die maximale Größe von »Warnboxen etc. festlegt). Falls Sie diesen Puffer benutzen wollen, müssen Sie verhindern, daß »AES diesen Puffer ebenfalls gebraucht (dies ge-

schieht mittels »*wind_update()*). So bekommen Sie die nötigen Angaben:

```
wind_get(0, WF_SCREEN, &loaddr, &hiaddr,
word     0,   WF_SCREEN;
long                          loaddr,   hiaddr;
                             &lolen, &hilen);
word *                        lolen,  hilen;
```

Bei den Langwörtern handelt es sich um die **Adressen** der Variablen - nicht um die Variablen selbst!

Nach diesem Aufruf enthalten 'loaddr' und 'hiaddr' die unteren bzw. die oberen 16 Bit der Pufferadresse, und 'lolen' bzw. 'hilen' die unteren 16 Bit bzw. die oberen 16 Bit der Pufferlänge (Sie müssen daraus das jeweils korrekte Langwort zusammensetzen).

Die Parameter, die mit dieser Routine abgefragt werden, kann man mit »*wind_set()* setzen - soweit dies sinnvoll ist.

Eingabeparameter:

```
word wi_ghandle     Fensternummer

word wi_gfield      Nummer der gewünschten Routine (s. o.)

word *wi_gw1
word *wi_gw2
word *wi_gw3
word *wi_gw4        Adressen der Rückgabeparameter (s. u.)

contrl(0)    =  104
contrl(1)    =    2
contrl(3)    =    0

int_in(0)    =  wi_ghandle
int_in(1)    =  wi_gfield
```

Ausgabeparameter:

```
word wi_greturn     0 = Ein Fehler ist passiert, n = alles Ok

word wi_gw1
word wi_gw2
word wi_gw3
```

word wi_gw4 Die Bedeutung dieser Parameter ist von dem Eingabeparameter wi_gfield abhängig - siehe Beschreibung oben!

```
contrl(2)   =   5
contrl(4)   =   0

int_out(0)  =  wi_greturn
int_out(1)  =  word wi_gw1
int_out(2)  =  word wi_gw2
int_out(3)  =  word wi_gw3
int_out(4)  =  word wi_gw4
```

Wind_open

Zur »Window Library gehörende Routine:

```
$65 = 101       wi_oreturn = wind_open(wi_ohandle, wi_ox, wi_oy,
word            wi_oreturn,            wi_ohandle, wi_ox, wi_oy;
                                       wi_oww, wi_owh)
word                                   wi_oww, wi_owh;
```

Mit dieser Routine kann ein »Fenster, welches zuvor mit der Funktion »*wind_create()* erzeugt wurde, auf dem Bildschirm dargestellt werden. D. h., genauer gesagt: der Rahmen des Fensters (für den im weiteren »AES zuständig ist) wird gezeichnet. Was im »Fenster dargestellt wird, ist Sache des Programms. Die Größe des »Fensters, die angegeben wird, ist daher die Größe des äußeren Rahmens. (Mit »*wind_get()* kann man herausbekommen, wie groß die Arbeitsfläche eines vorhandenen Fensters ist, mit »*wind_calc()* läßt sich dies auch für hypothetische Fenster berechnen).

Die Maße, die bei »*wind_create()* angegeben wurden, beziehen sich auf die maximale Größe des »Fensters, die hier angegebenen auf die Größe (und Position) des »Fensters beim Öffnen - es kann aber niemals größer sein als bei »*wind_create()* gesetzt.

Eingabeparameter:

word wi_ohandle Die (Handle-) Nummer des Fensters, die von der Routine »*wind_create()* zuvor zurückgegeben wurde (»Fensternummer)

word wi_ox X-Koordinate der linken oberen Ecke des Fensterrahmens

word wi_oy Y-Koordinate der linken oberen Ecke des Fensterrahmens

```
word wi_ow          Die Breite des Fensterrahmens in Pixeln

word wi_oh          Die Höhe des Fensterrahmens in Pixeln

contrl(0)   =  101
contrl(1)   =    5
contrl(3)   =    0

int_in(0)   =  wi_ohandle
int_in(1)   =  wi_owx
int_in(2)   =  wi_owy
int_in(3)   =  wi_oww
int_in(4)   =  wi_owh
```

Ausgabeparameter:

```
word wi_oreturn     0 = Ein Fehler ist passiert, n = alles Ok

contrl(2)   =    1
contrl(4)   =    0

int_out(0)  =  word wi_oreturn
```

Wind_set

Zur »Window Library gehörende Routine:

```
$69 = 105       wi_sreturn = wind_set(wi_shandle, wi_sfield,
word              wi_sreturn,           wi_shandle, wi_sfield:
                                        wi_s1, wi_s2, wi_s3, wi_s4)
word                                    wi_s1, wi_s2, wi_s3, wi_s4;
```

Diese Routine ist das Gegenstück zu »*wind_get()*, d. h., die Parameter, die mit »*wind_get()* abgefragt werden, werden mit dieser Routine gesetzt.

Die Routinennummer 'wi_sfield', die angibt, welche Fensterdaten mit der Routine geändert werden sollen, kann folgende Werte annehmen (die symbolischen Namen sind wieder "»GEMDEFS.H" entnommen):

```
 1 - WF_KIND        Hiermit können die Bestandteile des Fensters
                    gesetzt (oder verändert) werden, wie bei
                    »wind_create(), jedoch auch nachträglich.
                    wi_sw1   enthält die Bestandteile des Fensters
                             - siehe auch »wind_create().
```

2 - WF_NAME

Dient zum Setzen der Titelzeile des Fensters. Übergeben wird die Adresse eines Strings in 'wi_sw1' und 'wi_sw2' - z. B. so:

```
wind_set (wi_shandle, WF_NAME,
word      wi_shandle, WF_NAME;
          string, 0, 0);
byte *    string;
word              0, 0;
```

'string' ist ein Zeiger auf den Text, der in der Titelzeile dargestellt werden soll. ACHTUNG! Der Zeiger muß auf eine statische (bzw. globale) Variable zeigen, da »AES beim Neuzeichnen des Fensterrahmens den Zeiger benutzt, um auf den String zugreifen zu können - existiert dieser nicht mehr, so kann das System zusammenbrechen!

wi_sw1
wi_sw2 Diese beiden Worte enthalten zusammen den Zeiger

3 - WF_INFO

Wie bei WF_NAME wird damit (ebenfalls vor »*wind_open()*) die »Infozeile gesetzt bzw. nachträglich geändert. Was bei WF_NAME gesagt wurde, gilt auch hier!

5 - WF_CURRXYWH

Hiermit kann die Fenstergröße neu bestimmt werden, inklusive aller Fensterbestandteile wie Titelzeile etc. Parameter:

wi_sw1 X-Koordinate
wi_sw2 Y-Koordinate
wi_sw3 Breite
wi_sw4 Höhe

8 - WF_HSLIDE

Wie bei »*wind_get()* erläutert, nur wird hier die Position des horizontalen Schiebers gesetzt.

9 - WF_VSLIDE

Wie bei »*wind_get()* erläutert, nur wird hier die Position des vertikalen Schiebers gesetzt.

10 - WF_TOP

Damit kann das Fenster, dessen »Fensternummer mit 'wi_shandle' übergeben wurde, zum obersten (aktiven) Fenster gemacht werden - »AES erzeugt daraufhin eine »Nachricht für das Fenster ("WM_TOPPED").

14 - WF_NEWDESK

Mit dieser Routinennummer ist es möglich, den »Desktop-Hintergrund neu zu setzen. Dazu kreiere man ein »Objekt (z. B. mit dem Resource Construction Program), das natürlich (!) groß genug sein muß (mind. 640 * 381 Pixel), und übergebe einen Zeiger darauf, etwa so:

```
wind_set(0, WF_NEWDESK, objekt_adress, 0, 0);
word     0, WF_NEWDESK,                0, 0;
word *                   objekt_adress;
```

'objekt_adress' ist ein (32-Bit-) Zeiger auf das gewünschte Objekt (welcher mittels »*rsrc_gaddr()* bestimmt wurde) - und fortan benutzt »AES, wenn der »Desktop-Hintergrund neu gezeichnet werden muß, dazu das »Objekt (welches beliebig komplex sein darf, auch »Objekte vom Typ "G_PROGDEF" sind möglich!).

wi_sw1 Objekt-Baum, untere 16 Bit
wi_sw2 Objekt-Baum, obere 16 Bit
wi_sw3 Das Objekt, ab dem mit dem Zeichnen begonnen werden soll

15 - WF_HSLSIZE

Die Größe des horizontalen Schiebers relativ zur Größe des »Rollbalkens. Ein Wert von -1 bedeutet dabei, daß der Schieber eine minimale Größe hat, sonst gehen die Werte von 1 bis 1000 (1000 = maximale Größe).

wi_sw1 Schiebergröße

16 - WF_VSLSIZE

Wie bei WF_HSLSIZE, nur werden die Werte für den vertikalen Schieber gesetzt.

wi_sw1 Schiebergröße

Eingabeparameter:

word wi_shandle — Die »Fensternummer des Fensters, dessen Parameter geändert werden sollen.

word wi_sfield — Routinennummer, die bestimmt, welche Parameter verändert werden sollen - siehe oben.

word wi_sw1
word wi_sw2
word wi_sw3
word wi_sw4

Parameter, deren Bedeutung von wi_sfield abhängen

```
contrl(0)   =  105
contrl(1)   =    6
contrl(3)   =    0
```

Ausgabeparameter:

```
word wi_sreturn     0 = Ein Fehler ist geschehen, n = alles Ok.

contrl(2)   =    1
contrl(4)   =    0

int_out(0)  =  wi_sreturn
```

Wind_update

Zur »Window Library gehörende Routine:

```
$6B = 107       wi_ureturn = wind_update(wi_ubegend)
word            wi_ureturn,              wi_ubegend;
```

Mit dieser Routine kann erreicht werden, daß »AES die Kontrolle über den »Bildschirm oder über die »Maus an das Programm übergibt, bzw. das Programm kann die Kontrolle über diese Dinge wieder dem »AES übertragen. Ersteres wird vor allem dazu verwendet, um »Fenster neu zu zeichnen ("Update").

Wenn »AES mitgeteilt wird, daß ein »Fenster neugezeichnet werden soll, so wird der zum »Fenster gehörende Teil des »Bildschirms nicht von »AES verändert (etwa durch ein »Drop-Down-Menü). Auch »Warnboxen funktionieren nicht mehr!

Wenn »AES dazu aufgefordert wird, die Kontrolle über die »Maus abzugeben, so reagiert »AES nicht mehr auf Maus-Eingaben, d. h., es werden keine »Drop-Down-Menüs mehr aufgebaut oder beim Anklikken von Fenster-Bestandteilen irgendwelche Aktionen unternommen, der »Screen-Manager ist also nicht mehr aktiv.

Eingabeparameter:

```
word wi_ubegend     Eine Nummer für die Art der gewünschten Rou-
                    tine (Hinweis: die symbolischen Namen können
                    Sie in der Datei "GEMDEFS.H" finden):
                    0 - END_UPDATE      Window-Update beenden
                    1 - BEG_UPDATE      Window-Update beginnen
                    2 - END_MCTRL       »AES übernimmt wieder die
                                        Kontrolle über die »Maus
                    3 - BEG_MCTRL       Das Programm übernimmt die
                                        Mauskontrolle
```

```
contrl(0)   =  107
contrl(1)   =    1
contrl(3)   =    0
```

Ausgabeparameter:

word wi_ureturn 0 = Ein Fehler ist vorgekommen, n = alles in Ordnung.

```
contrl(2)   =    1
contrl(4)   =    0

int_out(0)  =  wi_ureturn
```

Window

Zu deutsch: Fenster. Bei W.'s handelt es sich um (rechteckige) Bildschirmausschnitte, die eingerahmt sind, und in denen der Computer Informationen darbietet (Text und/oder Grafik). In der Abbildung im Anhang A sehen Sie ein (voll ausgestattetes) Fenster, wie es »GEM bzw, »AES darstellt. Die verschiedenen Fensterbestandteile, die Sie in der Abbildung sehen können, müssen nicht alle bei Fenstern vorhanden sein. Ein Window besteht im einfachsten Fall nur aus einer mit einfachen Linien umrahmten Fläche, wobei die Linien Schatten werfen.

Meistens werden Fenster dazu benutzt, Daten darzustellen, die nicht ganz auf den »Bildschirm passen. Dann ist es meistens möglich, das Fenster über den Daten hin- und herzuschieben.

Die Bedeutung der Fensterteile (in Klammern der englische Ausdruck):

- Titelzeile (Titel) — In der Titelzeile finden Sie den Namen oder die Bezeichnung des Fensters
- Infozeile (Infoline) — Die Informationszeile ist für weitergehende Informationen über das Fenster gedacht
- Titelbalken (Movebar) — Mit diesem Balken kann das Fenster bewegt werden. Der Benutzer klickt es dazu mit der »Maus an, hält den Mausknopf gedrückt und bewegt dann mit der Maus einen Fensterumriß. Läßt er den Mausknopf los, so wird das Fenster an die entsprechende Stelle transportiert (sofern das Programm das erlaubt).
- Schließbox (Closebox) — Das Anklicken dieser Box signalisiert dem Programm, daß der Benutzer es schließen möchte.

- Vollbox (Fullbox) — Wenn das Fenster nicht auf maximaler Größe ist, so bewirkt ein Anklicken dieser Box, daß das Fenster sich auf maximal vorgesehene Größe ausdehnt. Wenn es bereits maximale Größe hat, so schrumpft es wieder auf die Ausdehnung, die es vorher hatte.

- Vergrößerungsbox (Sizebox) — Wenn man dieses Feld anklickt, den Mausknopf gedrückt hält, so kann dann mit der Maus die Größe des Fensters verändert werden (durch Verschieben der »Maus).

- Vertikaler Rollbalken (Vertical Scrollbar) — Durch Anklicken des grauen Balkens wird dem Programm der Befehl gegeben, den Fensterinhalt um eine Seite nach oben oder nach unten (je nachdem, ob es über dem Schieber oder darunter geschah) innerhalb des Fensters zu verschieben.
  Was eine Seite bedeutet, liegt an der Art der dargestellten Daten.

- Horizontaler Rollbalken (Horizontal Scrollbar) — Wie beim vertikalen Scrollbalken, nur wird in horizontaler Richtung verschoben.

- Rollpfeile (Arrows) — Ein Anklicken der Pfeile bewirkt, daß der Fensterinhalt in die entsprechende Richtung um eine Zeile verschoben wird.
  Was eine Zeile ist, wird durch die Art der dargestellten Daten festgelegt.

- Arbeitsfläche (Working Area) — Dies ist das eigentliche Arbeitsfeld, in dem die Daten dargestellt werden. Meistens sieht man durch das Fenster nur einen Teil der zum Fenster gehörenden Daten.

Die Bedeutung der einzelnen Fensterbestandteile ist durch Konvention festgelegt. Manche Programme interpretieren das Anklicken eines Fensterbestandteils auch anders.

Falls Sie Fenster programmieren wollen, so lesen Sie bitte folgende Stichworte:
»Window Library (und alle dort erwähnten Routinen, insbesondere aber »*wind_create()*)
»Window handle
»Event
»Event Library (und alle dort erwähnten Routinen)

»Clipping
»Window Update

Danach sollte es Ihnen keine (kaum) Probleme bereiten, Fenster zu programmieren.

Window handle

Engl., wörtlich übersetzt "Fenstergriff". Es handelt sich dabei um eine positive ganze Zahl (16 Bit), die den Zugriff auf ein bestimmtes Fenster erlaubt. Die Routine »*wind_create()* liefert diese Nummer zurück, wenn eine Fenster kreiert wird. Alle Routinen zur Manipulation von Fenstern (auch die, die Informationen über ein Fenster abfragen, mit Ausnahme von »*wind_calc()*) verlangen als ein Parameter die Nummer des Fensters, das angesprochen werden soll. Insgesamt können von Programmen bis zu acht Fenstern erzeugt werden.

Fensternummer Null bezieht sich auf ein spezielles Fenster: es handelt sich dabei um den Desktop-Hintergrund. Dieses Fenster kann behandelt werden wie jedes beliebige andere Fenster auch, allerdings kann es nicht in seiner Größe verändert oder gelöscht werden. Seine Fläche ist zugleich die maximal nutzbare Fläche für Fenster (damit noch Platz für die Menüleiste bleibt). Außerdem ist für das Neuzeichnen des Fensterinhalts »AES verantwortlich.

Window-Update

Fenster sind für den Benutzer eine angenehme Art, mit mehreren Informationen gleichzeitig umzugehen. Für den Programmierer bedeutet dies allerdings einen erheblich höheren Programmieraufwand - das ist der Preis des Fortschritts.

Was für viele Programmierer ein Buch mit sieben Siegeln ist - das Neuzeichnen von Fenstern - soll im weiteren erläutert werden.

Die meisten Anforderungen zum Neuzeichnen eines Fensters werden von »AES selbst erzeugt und erreichen das Programm in Form einer »Nachricht. Dabei muß die Anwendung AES mitteilen, daß sie auf eine »Nachricht wartet (dies geschieht meistens mit »*evnt_multi()*, kann jedoch auch mit »*evnt_mesag()* erreicht werden).

Bereiche des Bildschirms können durch Verschieben eines Fensters, Verkleinern eines Fensters, Verschwinden einer Dialogbox oder dem Schließen eines Fensters "zerstört" werden. Wenn »GEM ein Rechteck entdeckt, das neu gezeichnet werden muß, so prüft es, welche Fenster vorhanden sind, und schickt jedem Fenster, welches dieses Rechteck berührt, eine »Nachricht: die REDRAW-MESSAGE (siehe auch die Definition der Ereignisse, WM_REDRAW). Die Randzonen eines

Fensters (Titelbalken, Infozeile, Rollbalken etc.) werden von »AES selbst neu gezeichnet.

Nun ist die Anwendung selbst dafür verantwortlich, welche Aktionen daraufhin durchgeführt werden sollen. Andere Betriebssysteme führen dies selbst durch, indem sie den kompletten Bildschirminhalt im Ram zwischenspeichern. Muß der Inhalt erneuert werden, so wird der vorher gespeicherte Inhalt in das Fenster transferiert. Dies Zwischenspeichern kostet sehr viel Speicherplatz. Die GEM-Methode spart Speicherplatz, ist aber deutlich langsamer.

Zurück zur »Nachricht: »GEM schickt die X- Y-Koordinaten, die Breite und die Höhe des neuzuzeichnenden Bereichs, aber »GEM prüft **nicht** nach, ob die Anwendung in diesem Bereich auch schadlos zeichnen darf! Die Anwendung muß dies selbst mithilfe der "Rechteckliste" herausfinden. Diese Liste wird von »GEM geführt und enthält diejenigen Fensterbereiche, die zuoberst liegen (nicht verdeckt sind), und in die gefahrlos hineingezeichnet werden kann.

Nehmen wir mal an, auf dem Bildschirm sind zwei Fenster, die sich überlappen. Wenn nun der Programmteil, der für das Neuzeichnen des unteren Fensters verantwortlich ist, dies ohne Rücksicht tut, so wird das obere Fenster teilweise vernichtet. Je nachdem, wie die Lage der beiden Fenster zueinander ist, wird das untere Fenster von dem oberen in ein oder mehrere Rechtecke zerteilt. Sehen Sie sich die entsprechenden Abbildungen im Anhang A an. Dort sehen Sie unser zwei-Fenster-Beispiel und die verschiedenen Möglichkeiten, die es gibt, sowie die Rechtecke, die in der GEM-Rechteck-Liste stehen, und in die beim Neuzeichnen des unteren Fensters gezeichnet werden darf. Mit mehr als zwei Fenstern kann das Ganze nahezu beliebig verkompliziert werden.

Bedingt durch die Anzahl der Möglichkeiten, wie die Rechtecke verteilt sein können, generiert »GEM nicht für jedes Rechteck eine eigene »Nachricht: sehr schnell würden sonst eine Flut von Nachrichten erzeugt werden müssen.

Für jedes Fenster existiert nun eine Liste von Rechtecken, in die das Programm zeichnen darf. Dies ist der Fensterbereich, der durch nichts überdeckt wird, also sichtbar ist. Diese Rechteckliste wird von »GEM so berechnet, daß die Anzahl der Rechtecke so gering wie möglich ist.

Wenn nun für ein Fenster die »Nachricht "WM_REDRAW" gesandt wird, so liest das Programm die Rechteckliste und zeichnet in diesen Rechtecken den Fensterinhalt neu.

Nehmen wir also an, Ihr Programm wartet mit einem »*evnt_multi()*-Aufruf auf eine »Nachricht. Ein Parameter für »*evnt_multi()* ist die Adresse eines 16 Byte großen Nachrichtenpuffers. Msg[0] sei das erste, Msg[1] das zweite Wort des Puffers usw. Dann steht in Msg[0] bei Rückkehr vom »*evnt_multi()*-Aufruf die Art der »Nachricht, sofern »AES eine sendet. Lautet die Nachricht "WM_REDRAW" (siehe »Message), so muß also (mindestens) ein Fenster neu gezeichnet werden. Die »Fensternummer steht in Msg[3], das neu zu zeichnende Rechteck in Msg[4] bis Msg[7]. In diesem Rechteck liegt (mindestens) ein Teil des Fensters. Das Fenster kann aber in diesem Bereich auch von anderen Fenstern überlappt sein.

Anmerkung: im folgenden Beispiel ist alles, was nicht **fett** gedruckt ist, als Kommentar zu betrachten. Wie immer benutze ich eine imaginäre Programmiersprache, die aber gewisse Ähnlichkeiten zu C und Pascal nicht verhehlen kann. Es empfiehlt sich, **jede** der erwähnten Funktionen nachzuschlagen, lediglich *redraw_window()* und *is_overlapped()* werden Sie nicht finden: die werden hier erklärt.

Also rufen wir eine Routine auf, die das Fenster neu zeichnen soll:

```
        redraw_window(w_handle, x, y, w, h)
word                  w_handle, x, y, w, h;
```

Diese Beispielroutine soll ein zerstörtes Fenster wieder restaurieren.

Eingabeparameter:

word w_handle	Die »Fensternummer des Fensters, das neu gezeichnet werden soll - enthalten in Msg[3]
word x	X-Koordinate des Bereichs, der zerstört worden ist - enthalten in Msg[4]
word y	Y-Koordinate des Bereichs, der zerstört worden ist - enthalten in Msg[5]
word w	Breite des Bereichs, der zerstört worden ist - enthalten in Msg[6]
word h	Höhe des Bereichs, der zerstört worden ist enthalten in Msg[7]

Ausgabeparameter: keine

Beginn von *redraw_window()*:

Definition lokaler Variablen:

```
word x_rect
word y_rect
word h_rect      Variablen für das jeweils neu zu zeichnende
word w_rect      Rechteck
```

```
        graf_mouse(256, 0)
word            256;
long                0;
```

Mit diesem Aufruf wird der »Mauszeiger abgeschaltet. Denn: der Hintergrund des Mauszeigers wird zwischengespeichert. Wird die Maus bewegt, so wird der Hintergrund wieder restauriert. Wurde einfach über den Mauszeiger gezeichnet, so wird der falsche (alte) Hintergrund wieder auf den »Bildschirm gepackt.

```
        wind_update(BEG_UPDATE)
word                BEG_UPDATE;
```

Diese Routine bitte unter »*wind_update()* nachschlagen! Sie ist notwendig, damit »AES den »Bildschirm nicht verändert.

Nun holen wir uns das erste Rechteck aus der Rechteckliste. In dieser Liste speichert der Window-Manager von »AES die Teile des Fensters, in die gefahrlos hineingezeichnet werden kann:

```
        wind_get(w_handle, WF_FIRSTXYWH, &x_rect, &y_rect,
word            w_handle, WF_FIRSTXYWH;
word *                                   x_rect,  y_rect,
                                         &w_rect, &h_rect)
word *                                   w_rect,  h_rect;

word w_handle    Oben erklärt

word WF_FIRSTXYWH  Diese Konstante (deren Wert elf beträgt) teilt
                   der Routine »*wind_get()* mit, daß wir das erste
                   Rechteck aus der Rechteckliste haben wollen.

word *x_rect
word *y_rect
word *w_rect       Dies sind die Adressen der weiter oben dekla-
word *h_rect       rierten lokalen Variablen. Nach Aufruf von
                   »*wind_get()* enthalten sie als Rückgabewerte
```

die Koordinaten eines Rechtecks aus der Rechteckliste

```
while(w_rect <> 0 AND h_rect <> 0)
```

Solange wir für die Breite und die Höhe von »*wind_get()* noch Werte bekommen, die ungleich Null sind, ist in der Liste noch ein Rechteck für uns. Also beginnen wir eine while-Schleife:

```
BEGIN_WHILE

        IF (overlap(full_x, full_y, full_w, full_h,
word                full_x, full_y, full_w, full_h;
                    x_rect, y_rect, w_rect, h_rect)
word                x_rect, y_rect, w_rect, h_rect)
        THEN
```

```
word full_x
word full_y
word full_w
word full_h
```

In diesen Variablen stehen die Werte für den ganzen »Bildschirm, z. B. für den monochromen Monitor sind dies die Werte 0, 19, 640 und 400.

Die Routine "overlap" berechnet, ob das angegebene Rechteck (in 'x_rect' etc. stehend) auch tatsächlich sich auf dem »Bildschirm befindet (davon kann man nämlich nicht ausgehen!). Nur wenn das der Fall ist, brauchen wir das Fenster neu zu zeichnen; was außerhalb des »Bildschirms vor sich geht, interessiert uns nicht. Wie diese Routine aussieht, überlassen wir Ihnen - die Routine soll eine Eins zurückgeben, wenn der Bereich, den die letzten vier Parameter beschreiben, innerhalb des von den ersten vier Parametern eingegrenzten Bereichs liegt, sonst eine Null. Auch wenn beide Bereiche identisch sind, sollte eine Eins zurückgegeben werden. Weiter:

```
IF(overlap(x, y, w, h,
           x_rect, y_rect, w_rect, h_rect)
   THEN
```

Nun muß getestet werden, ob sich der Bereich aus der Rechteckliste in 'x_rect' etc. auch innerhalb des "zerstörten" Bereichs auf dem Schirm befindet. Ist das nicht der Fall, so braucht auch nichts unternommen zu werden (der "zerstörte Bereich steht in 'x', 'y', 'w' und 'h'). Wenn dies der Fall ist, muß das mit 'w_handle' bestimmte Fenster in dem durch 'x_rect' etc. festgelegten Bereich neu gezeichnet werden.

```
IF(w_handle = first_window)
THEN
```

An dieser Stelle steht nun der Aufruf der Routine, die den (gesamten) Fensterinhalt für das erste Ihrer Fenster neu zeichnen kann (first_window ist die »Fensternummer Ihres ersten Fensters, 'second_window' die Ihres zweiten usw.). WICHTIG: setzen Sie mit »*vs_clip()* den Bereich, in dem gezeichnet wird. Das ist in diesem Fall das durch 'x_rect', 'y_rect', 'w_rect' und 'h_rect' definierte Rechteck. Denn nur in diesem Rechteck dürfen Sie ja gefahrlos zeichnen!

Die hier aufgerufene Code darf **nur** Routinen benutzen, die »Clipping beherrschen! Also »VDI- »AES- oder »GEM-Routinen.

```
ENDIF
```

Nun testen Sie einfach nacheinander für Ihre Fenster ab, ob sie gemeint sind:

```
IF(w_handle = second_window)
THEN
...
ENDIF
```

Dies machen Sie für jedes Ihrer Fenster.

```
ENDIF
```

Das obige 'ENDIF' beendet den Block "IF(overlap(x, y, w, ... usw."

```
ENDIF
```

Dies beendet IF(overlap(full_x, ... usw.

Die While-Schleife muß nun für jedes Rechteck aus der Rechteckliste nochmal durchlaufen werden, solange, bis kein Rechteck mehr vorhanden ist (die Höhe und Breite des Rechtecks, dessen Werte »*wind_get()* zurückliefert, sind dann Null). Folgender Routinenaufruf holt das nächste Rechteck:

```
wind_get(w_handle, WF_NEXTXYWH, &x_rect, &y_rect,
                                &w_rect, &h_rect)
```

Die Konstante 'WF_NEXTXYWH' ist gleich zwölf. Nun sind wir am Ende der While-Schleife:

```
END_WHILE
```

```
        wind_update(END_UPDATE)
word                END_UPDATE;
```

Diese Routine bitte unter »*wind_update()* nachschlagen!

```
        graf_mouse(257, 0)
word                257;
long                    0;
```

Der »Mauszeiger muß wieder sichtbar gemacht werden.

Damit ist die Routine *redraw_window()* beendet! Weil die Beschreibung durch ihre Ausführlichkeit etwas unübersichtlich ist, folgt sie im Zusammenhang:

```
      redraw_window(w_handle, x, y, w, h)
      BEGINN

         graf_mouse(256, 0)
         wind_update(BEG_UPDATE)

         wind_get(w_handle, WF_FIRSTXYWH, &x_rect, &y_rect,
                                          &w_rect, &h_rect)

         while(w_rect <> 0 AND h_rect <> 0)

            BEGIN_WHILE

            IF (overlap(full_x, full_y, full_w, full_h,
                        x_rect, y_rect, w_rect, h_rect)
               THEN

                  IF(overlap(x, y, w, h,
                             x_rect, y_rect, w_rect, h_rect)
                     THEN

                        IF(w_handle = first_window)
                        THEN

HIER STEHT DER CODE ZUM NEUZEICHNEN IHRES ERSTEN FENSTERS MIT BE-
RÜCKSICHTIGUNG VON »CLIPPING!

                        ENDIF

                        IF(w_handle = second_window)
                        THEN

HIER STEHT DER CODE ZUM NEUZEICHNEN IHRES ZWEITEN FENSTERS MIT
BERÜCKSICHTIGUNG VON »CLIPPING!
```

```
                    ENDIF
```

Dies machen Sie für jedes Ihrer Fenster.

```
                ENDIF

            ENDIF

        wind_get(w_handle, WF_NEXTXYWH, &x_rect, &y_rect,
                                        &w_rect, &h_rect)

        END_WHILE

        wind_update(END_UPDATE)
        graf_mouse(257, 0)
    END
```

ENDE von *redraw_window()*

Workstation

Siehe »virtual Workstation und »*v_opnvwk()* sowie »*v_opnwk()*

XBIOS
Siehe »Extended Bios

Zeichensatz
Siehe »Font

Zweierkomplement
Das Z. ist eine Art der Zahlendarstellung. Im Dezimalsystem gibt es 10 Ziffern. Damit lassen sich beliebig große positive ganze Zahlen darstellen. Um gebrochen-rationale Zahlen zu erzeugen, benötigt man ein weiteres Zeichen, den Dezimalpunkt. Zur Darstellung negativer Zahlen wird noch ein neues Zeichen benötigt, das Minuszeichen '-'. In einem Computer gibt es jedoch normalerweise nur zwei Zustände, Null oder Eins. Mit diesen beiden Ziffern müssen alle Zahlen erzeugt werden können, auch negative. Man benutzt das höchstwertigste Bit der Zahl als Vorzeichen. Ist es gesetzt, ist die Zahl negativ, sonst positiv. Dies gilt sowohl für das Einer- als auch für das Zweierkomplement.

Man kehrt das Vorzeichen einer Zahl im Z. um, indem man alle Bits invertiert (das gibt das Einerkomplement) und dann noch Eins addiert. Hier ein Beispiel:

```
                        %00001010 = 10
Bits invertieren:   --> %11110101
Eins addieren:      --> %11110110 = -10
```

Anhang A

Die Bilder im diesem Anhang beziehen sich auf die entsprechenden Verweise im Text!

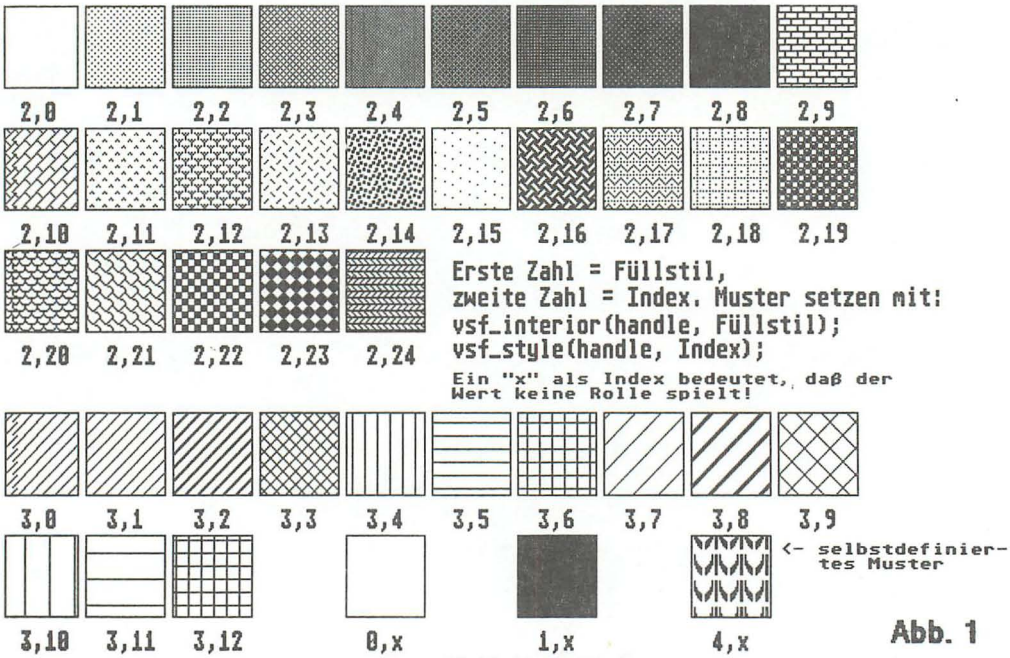

2,0 2,1 2,2 2,3 2,4 2,5 2,6 2,7 2,8 2,9
2,10 2,11 2,12 2,13 2,14 2,15 2,16 2,17 2,18 2,19
2,20 2,21 2,22 2,23 2,24
Erste Zahl = Füllstil,
zweite Zahl = Index. Muster setzen mit:
vsf_interior(handle, Füllstil);
vsf_style(handle, Index);
Ein "x" als Index bedeutet, daß der Wert keine Rolle spielt!
3,0 3,1 3,2 3,3 3,4 3,5 3,6 3,7 3,8 3,9
<- selbstdefinier-tes Muster
3,10 3,11 3,12 0,x 1,x 4,x

Abb. 1

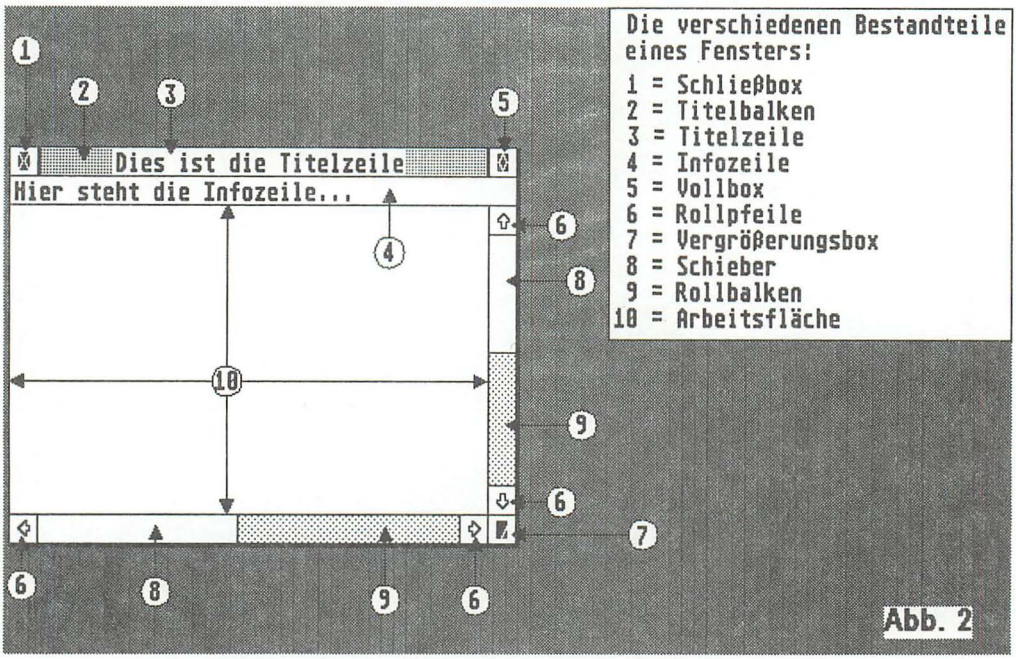
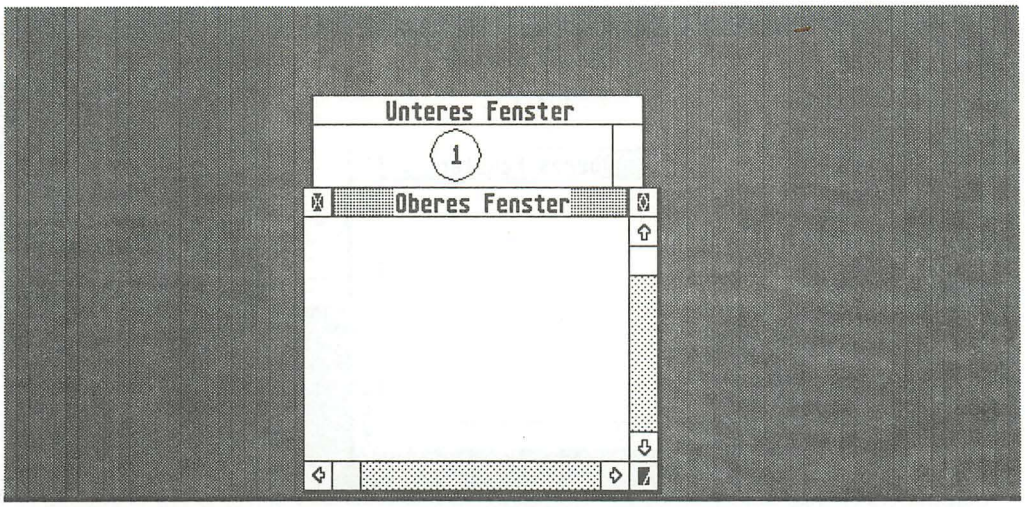
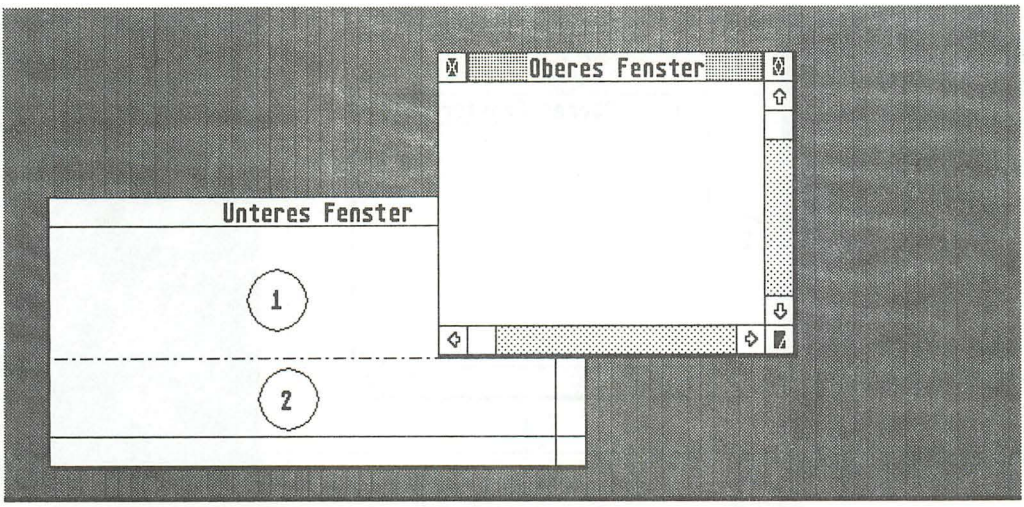

Dies ist die Titelzeile
Hier steht die Infozeile...
Die verschiedenen Bestandteile eines Fensters:
1 = Schließbox
2 = Titelbalken
3 = Titelzeile
4 = Infozeile
5 = Vollbox
6 = Rollpfeile
7 = Vergrößerungsbox
8 = Schieber
9 = Rollbalken
10 = Arbeitsfläche

Abb. 2



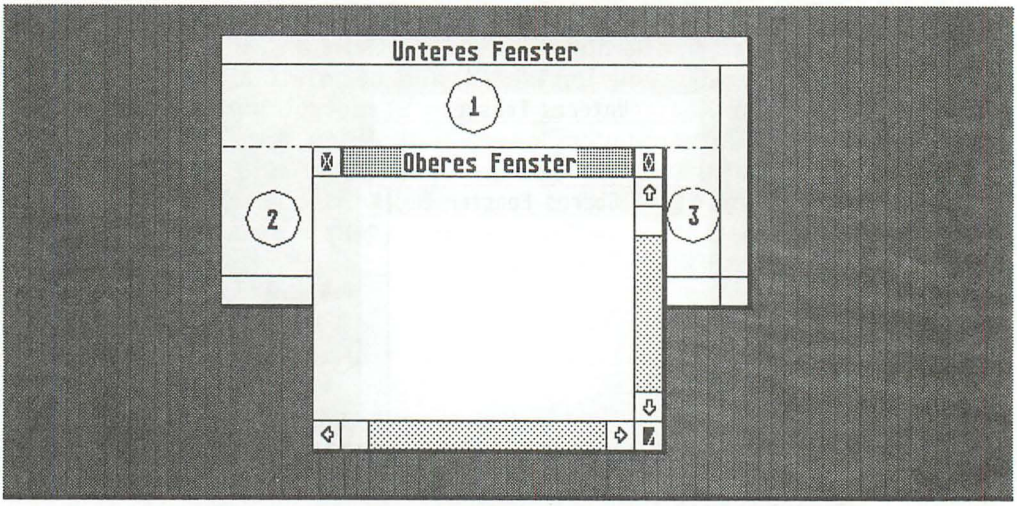
Durch Überlappen entsteht im unteren Fenster ein Rechteck.
Nur in dieses darf gezeichnet werden.

Abb. 3



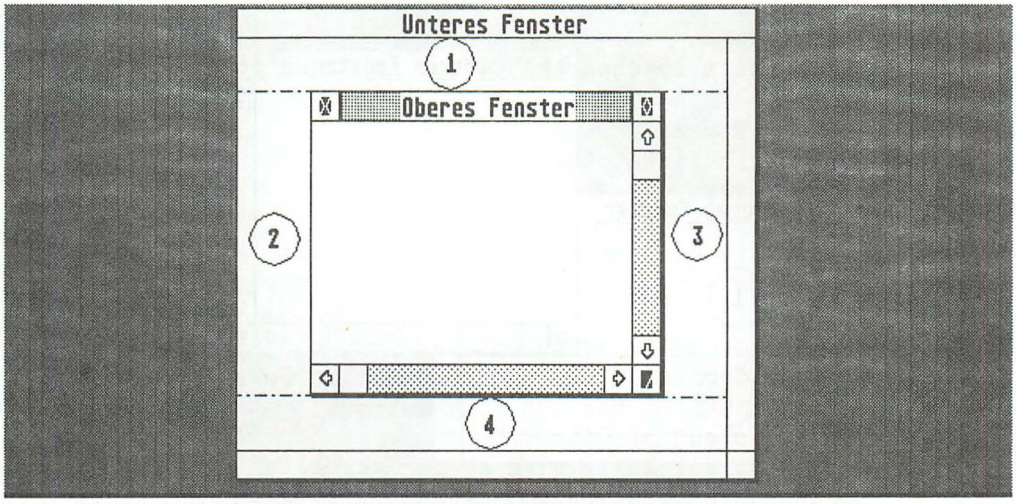
Durch Überlappen enststehen im unteren Fenster zwei Rechtecke.
Nur in diese Rechtecke darf gezeichnet werden, ohne etwas zu zerstören.

Abb. 4



Durch Überlappen enststehen im unteren Fenster drei Rechtecke.
Nur in diese Rechtecke darf gezeichnet werden, ohne etwas zu zerstören.

Abb. 5



Durch Überlappen enststehen im unteren Fenster vier Rechtecke.
Nur in diese Rechtecke darf gezeichnet werden, ohne etwas zu zerstören.

Abb. 6

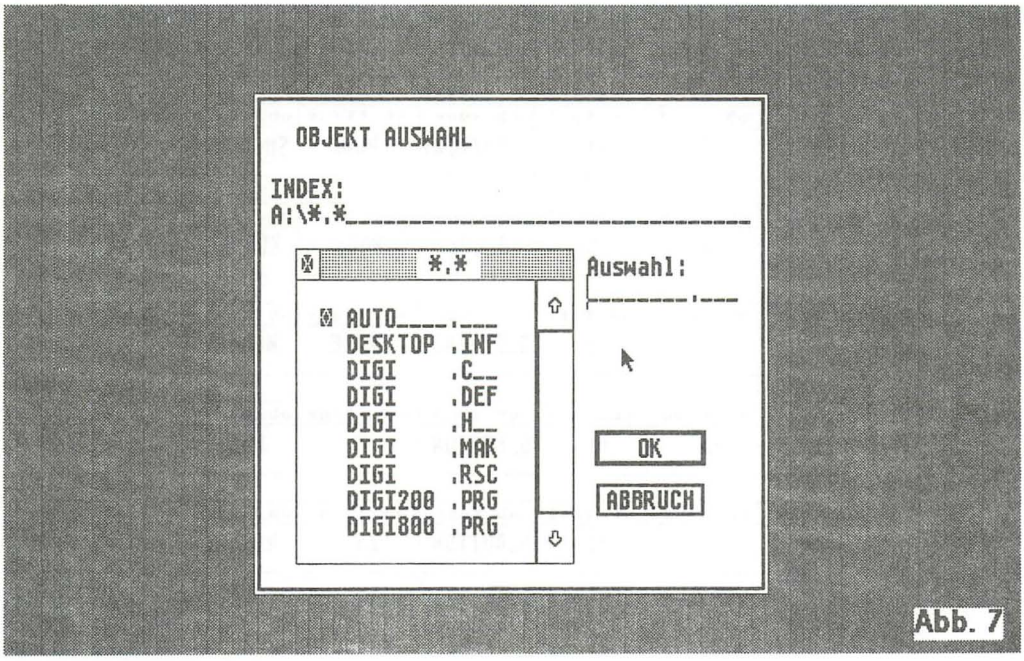

OBJEKT AUSWAHL
INDEX:
A:*.*
.
AUTO____.___
DESKTOP .INF
DIGI .C__
DIGI .DEF
DIGI .H__
DIGI .MAK
DIGI .RSC
DIGI200 .PRG
DIGI800 .PRG
Auswahl:
OK
ABBRUCH

Abb. 7

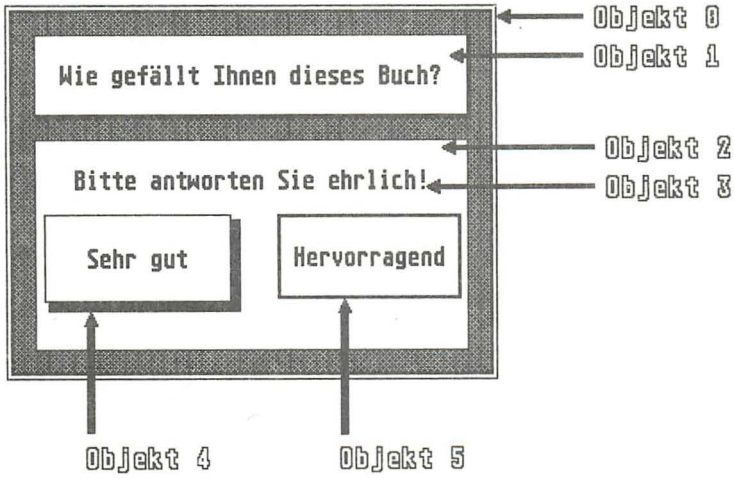

Wie gefällt Ihnen dieses Buch?
Bitte antworten Sie ehrlich!
Sehr gut
Hervorragend
Objekt 0
Objekt 1
Objekt 2
Objekt 3
Objekt 4
Objekt 5

Abb. 8

Indexnr.	ob_next	ob_head	ob_tail	ob_type	ob_flags	ob_state	ob_spec	...
0	-1	1	2	G_BOX	NONE	OUTL.	$21151	
	ob_next	ob_head	ob_tail	ob_type	ob_flags	ob_state	ob_spec	...
1	2	-1	-1	G_BOXTEXT	NONE	SHADOWED	0	
	ob_next	ob_head	ob_tail	ob_type	ob_flags	ob_state	ob_spec	...
2	0	3	5	G_BOX	NONE	NORMAL	$FF1100	
	ob_next	ob_head	ob_tail	ob_type	ob_flags	ob_state	ob_spec	...
3	4	-1	-1	G_STRING	NONE	NORMAL	3	
	ob_next	ob_head	ob_tail	ob_type	ob_flags	ob_state	ob_spec	...
4	5	-1	-1	G_BUTTON	7	$30	4	
	ob_next	ob_head	ob_tail	ob_type	ob_flags	ob_state	ob_spec	...
5	2	-1	-1	G_BUTTON	25	NORMAL	5	

Abb. 9

Anhang B

Im folgenden finden Sie ein paar Beispielprogramme. Diese bewirken alle dasselbe und sind in mehreren Programmiersprachen geschrieben.

Die Programme demonstrieren die sagenumwobene »GEM-Programmierung einer »Alarmbox, also etwas sehr Einfaches. Am schönsten sieht das Ergebnis in Assembler aus, aber Sie werden lachen, wir haben es sogar ausprobiert (und es hat sogar funktioniert... ehrlich!).

Jedes der Programme zeigt Ihnen viermal hintereinander eine »Alarmbox, jedesmal mit einem anderen »Icon.

Als erstes beschreiben wir die Programmierung in Omikron-Basic. Dazu muß noch angemerkt werden, daß hinter unser Beispiel noch eine »Library gehört, die wir hier nicht mit abdrucken. Sie gehört aber zum Lieferumfang des Basic dazu.

Omikron-Basic:

```
100 Appl_Init                        'Anmelden der »Application
110 Form$="[1][Dies ist eine ¦ Warnbox!][ Aha ]" 'Boxmeldung
120 FOR I=0 TO 3                     'Viermal ausgeben
130 MID$ (Form$,2,1)= CHR$(I+48)     'Anderes Icon einstellen
140 Form_Alert(1,Form$,Button)       'Alarmbox ausgeben
150 NEXT I                           'Schleifenende
160 Appl_Exit                        'Application abmelden
170 END                              'Programmende
```

Jetzt kommt das Beispiel in GFA-Basic. Man könnte das ganze auch erheblich einfacher haben (ALERT-Befehl), aber hier wollten wir ganz allgemein demonstrieren, wie sich auch andere (nicht implementierte) »GEM-Befehle programmieren lassen:

GFA-Basic:

```
F$="[0][ Dies ist eine ¦ Warnbox!][ Aha ]" 'Boxmeldung
For I=0 To 3                         'Viermal ausgeben
  Mid$(F$,2,1)=Chr$(I+48)            'Anderes Icon einstellen
  @Form_alert(1,F$)                  'Prozedur Form_alert aufrufen
Next I                               'Schleifenende
End                                  'Ende des Programms
Procedure Form_alert(N,A$)           'Prozedur Form_alert
  Dpoke Gcontrl,52                   'AES-Array besetzen (Opcode)
  Dpoke Gcontrl+2,1                  'Anzahl int_in-Werte
  Dpoke Gcontrl+6,1                  'Anzahl addr_in-Werte
  Dpoke Gintin,N                     'Default-Button
  Lpoke Addrin,Varptr(A$)            'Adresse des Strings
  Gemsys                             'Aufruf
  Button=Dpeek(Gintout)              'Rückgabewert
Return                               'Ende
```

Jetzt das ganze nochmal, weil's so schön war - aber in Assembler. Was bei diesem Beispiel weggelassen wurde, ist die Verwendung von *Mshrink()* (siehe »GEMDOS), um den nicht benötigten Speicher freizugeben.

Assembler:

```
            bsr     appl_init   * Application anmelden
            move.l  #3,d1       * Schleifenzähler
            move.w  #48,d2      * Nummer des Icon
            move.w  #1,d0       * Default-Button
loop:
            lea     form_string,a0 * Adresse des Strings
            move.b  d2,1(a0)    * Icon einstellen
            movem.l d0-d2,-(a7)* Register retten

            bsr     form_alert * Warnbox ausgeben

            movem.l (a7)+,d0-d2* Register zurückholen
            addq.b  #1,d2       * Nächstes Icon
            dbf     d1,loop     * Weiter im Schleifentext

ende:
            clr.w   -(a7)       * Funktionsnummer für Ende
            trap    #1          * Pterm0() ausführen


appl_init:
            lea     control,a1 * Adresse des control-Arrays
            move.w  #10,0(a1)  * Funktionsnummer laden
            move.w  #0,2(a1)   * Länge int_in-Array
            move.w  #0,6(a1)   * Länge addr_in-Array

            bsr     aes        * AES-Funktion aufrufen

            move.w  int_out,d0 * Anmeldung erfolgreich?
            bmi     ende       * Nein: Abbruch

            rts                * Ende

form_alert:
* D0 enthält die Nummer des Default-Buttons
* A0 enthält Zeiger auf String
```

```
                lea     control,a1 * Adresse des control-Arrays
                move.w  #52,0(a1)  * Funktionsnummer
                move.w  #1,2(a1)   * Länge des int_in-Arrays
                move.w  #1,6(a1)   * Länge des addr_in-Arrays

                move.w  d0,int_in  * Default-Button
                move.l  a0,addr_in * Adresse des

                bsr     aes        * AES aufrufen

                move.w  int_out,d0 * Rückgabewert
                rts                * Ende


aes:
                move.l  #Aes_paramblk,d1 * Adresse des AES-Para-
                                         * meterblocks
                move.w  #$c8,d0    * Kennummer für AES-Funktion
                trap    #2         * GEM aufrufen
                rts                * Ende

Aes_paramblk:                      * AES-Parameterblock
                dc.l    control    * Verschiedene Arrays
                dc.l    global
                dc.l    int_in
                dc.l    int_out
                dc.l    addr_in
                dc.l    addr_out

control:        ds.w    10         * Längen der Arrays:
global:         ds.w    15         * siehe »AES
int_in:         ds.w    128
int_out:        ds.w    128
addr_in:        ds.l    10
addr_out:       ds.l    10

form_string:                       * Text für Warnbox
                dc.b    '[0][Dies ist eine|Warnbox!][ Aha ]',0
```

Als letztes das Beispiel in C.

C:

```
int contrl[12],     /* Definition der Arrays    */
    intin[128],     /* Alle etwas groß, aber... */
    intout[128],    /* ... macht nix            */
    ptsin[128],
    ptsout[128];
```

```
main()                  /* Hauptprogramm          */
{
      int i, button;                /* Variablendeklaration    */
      static char form_string[] =
      "[0][Dies ist eine|Warnbox!][ Achwas ]";

      if (appl_init()) < 0)         /* Falls Anmeldung der App- */
         exit ();                   /* lication scheitert:      */
                                    /* Abbruch                  */
      for (i = 0; i < 4; i++)       /* Vier Schleifendurchläufe */
         {
         form_string[1] = (char) i + '0';   /* Anderes Icon    */
         button = form_alert (1, form_string);
         }                          /* Warnbox ausgeben         */
                                    /* Geschafft !              */
}
```

Natürlich kann man alle diese Programme noch eleganter, schneller und besser schreiben, weniger umständlich und überhaupt... Es sind halt nur Beispiele.

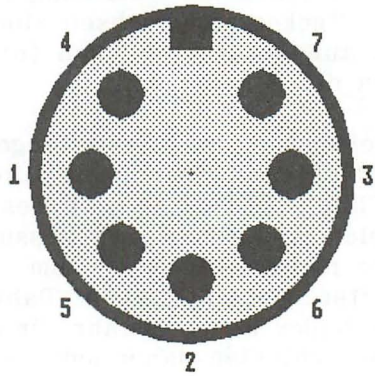
Anhang C

Auf den folgenden Seiten sehen Sie Abbildungen zu den Schnittstellen des Atari. Alle Stecker und Buchsen sind stets so zu sehen, als wenn Sie von **außen** darauf blicken (**nicht** innen vom Rechner her). Dann stimmen die Belegungen.

Eventuelle Unähnlichkeiten mit den Darstellungen im Atari-Handbuch sind **volle Absicht!** Diese Zeichnungen wurden vom einem begabten Elektroniker (Thies Hammerich) nach bestem Wissen und Gewissen erstellt, angereichert durch eigene Messungen (soweit möglich). Es läßt sich aber nicht (aller Mühe zum Trotz) aussschließen, daß hier noch weitere Fehler stecken! Daher garantieren wir für **nichts**. Wir können leider keine Gewähr für die Richtigkeit der Belegung geben und schließen daher jede Haftung für Schäden jeder Art aus, die durch unrichtige Angaben verursacht wurden. Wir haben Sie gewarnt!

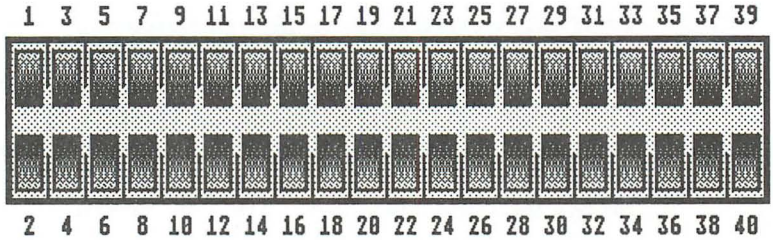
Die Abkürzung "NC" steht übrigens für "Not Connected" = nicht verbunden.

Power-Stecker



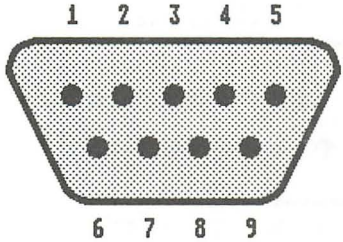
1— + 5 Volt DC
2— NC
3— Masse
4— + 5 Volt DC
5— + 12 Volt DC
6— - 12 Volt DC
7— Masse

Cartridge (ROM-Modul)



1—DC + 5 Volt
2—DC + 5 Volt
3—Data 14
4—Data 15
5—Data 12
6—Data 13
7—Data 10
8—Data 11
9—Data 8
10—Data 9
11—Data 6
12—Data 7
13—Data 4
14—Data 5
15—Data 2
16—Data 3
17—Data 0
18—Data 1
19—Adressltg. 13
20—Adressltg. 15
21—Adressltg. 8
22—Adressltg. 14
23—Adressltg. 7
24—Adressltg. 9
25—Adressltg. 6
26—Adressltg. 10
27—Adressltg. 5
28—Adressltg. 12
29—Adressltg. 11
30—Adressltg. 4
31—ROM-Select 3
32—Adressltg. 3
33—ROM-Select 4
34—Adressltg. 2
35—Upper DataStrobe
36—Adressltg. 1
37—Lower DataStrobe
38—
39— } Masse
40—

Joystick-/Maus-Stecker



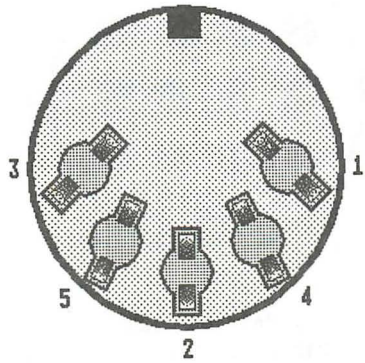
Stecker 0

1—Hoch / XB
2—Runter / XA
3—Links / YA
4—Rechts / YB
5—Reserviert (NC)
6—Feuer / Linke Maustaste
7— + 5 Volt DC
8—Masse
9—Feuer1 / Rechte Maustaste

Stecker 1

1—Hoch
2—Runter
3—Links
4—Rechts
5—Reserviert
6—Feuer (= Anschluß 9, Stecker0)
7— + 5 Volt DC
8—Masse
9—NC

Midi-Buchsen



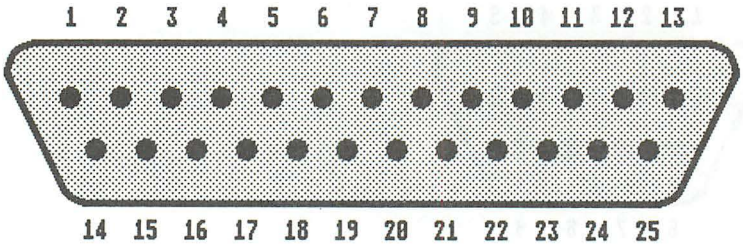
Midi-IN

1--NC
2--NC
3--NC
4--IN receive data
5--IN Loop Return

Midi-OUT

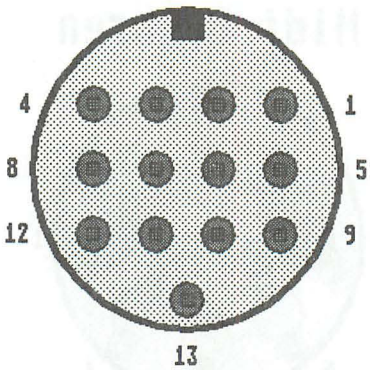
1--THRU Loop Return
2--Masse
3--THRU transmit data
4--OUT Loop Return
5--OUT transmit data

Modem-Stecker



1-- Schutzerde
2-- Sendedaten (TxD)
3-- Empfangsdaten (RxD)
4-- RTS-Signal (Request to send)
5-- CTS-Signal (Clear to send)
6-- NC (sonst:Data set ready DSR)
7-- Signal-Masse
8-- DCR-Signal (Carrier-detect, DCD)
9-- bis
19-- NC
20-- DTR-Signal (Data Terminal ready)
21-- NC
22-- ring indicator (RI)
23-- NC
24-- NC
25-- NC

Monitor-Buchse

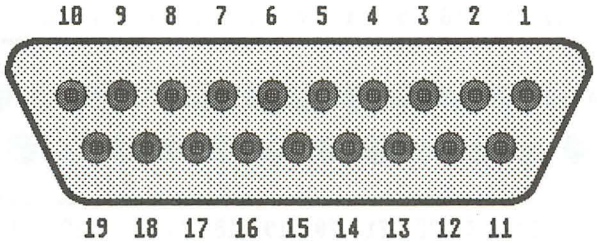


1--Audio Ausgang
2--Composite Sync.*)
3--allg. Ausgang
4--Monochrom-Sensor
5--Audio-Eingang
6--Grün
7--Rot
8-- + 12 Volt DC (10mA)**)
9--Horizontal Sync.
10--Blau
11--Monochrom-Signal
12--Vertikal Sync.
13--Masse

*) fehlt eventuell bei älteren Geräten

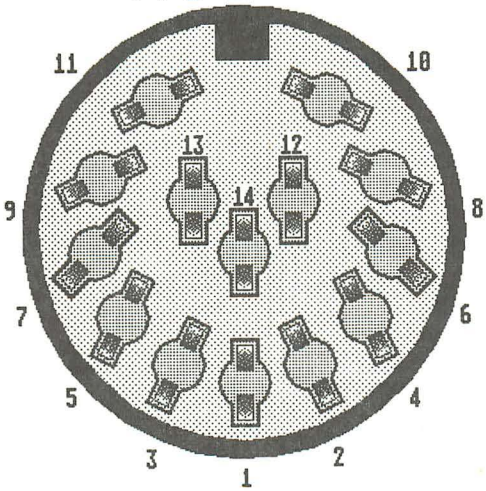
**) bei einigen älteren Geräten eventuell Masse

Harddisk-Buchse



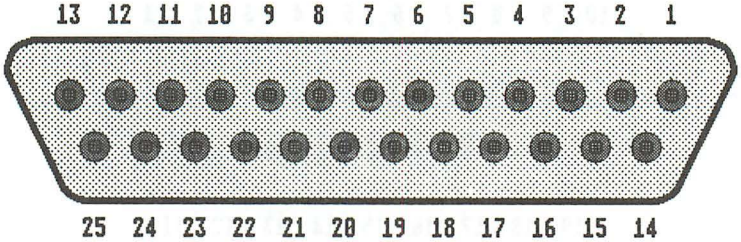
1—Data 0
2—Data 1
3—Data 2
4—Data 3
5—Data 4
6—Data 5
7—Data 6
8—Data 7
9—Select
10—IR Interrupt Anforderung
11—Masse
12—Rücksetzen
13—Masse
14—Bestätigungssignal (Ackn)
15—Masse
16—A 1
17—Masse
18—R/W Lesen/Schreiben
19—DR Daten-Anforderung

Floppy-Anschluß



1-read data
2-side 0 select
3-Masse
4-index
5-drive select 0
6-drive select 1
7-Masse
8-motor on
9-direction select
10-step
11-write data
12-write gate
13-track 00
14-write protect

Printer-Buchse



1—Data Strobe	11—Busy	21—Masse
2—Data 0	12—NC	22—Masse
3—Data 1	13—NC	23—Masse
4—Data 2	14—NC	24—Masse
5—Data 3	15—NC	25—Masse
6—Data 4	16—NC	
7—Data 5	17—NC	
8—Data 6	18—Masse	
9—Data 7	19—Masse	
10—NC	20—Masse	

Anhang D

Alle Angaben im folgenden beziehen sich auf die **deutsche Tastatur!**

In diesem Anhang finden Sie eine komplette Tastaturtabelle. Sehen Sie sich bitte die Abbildungen eins und zwei in diesem Anhang an. Abb. 1 zeigt den Schreibmaschinen-ähnlichen Hauptblock der Tastatur. Oben sind die Funktionstasten zu erkennen. In dem Rechteck auf den Tasten sehen Sie den sog. »Scancode (siehe entsprechendes Stichwort). Auf den Funktionstasten finden Sie zwei (hexadezimale) Ziffern: der obere »Scancode ergibt sich durch gleichzeitiges Drücken der <Shift>-Taste und der Funktionstasten. Bei **allen** anderen Tasten ändert sich der »Scancode durch Drücken der <Shift>-Taste **nicht**. Ob eine Funktionstaste gedrückt wurde, läßt sich folglich nur durch Testen des »Scancodes herausfinden.

Neben dem »Scancode liefert jede Taste noch einen »ASCII-Code (siehe »Scancode).

Im weiteren finden Sie eine vollständige Tastaturtabelle. Diese wurde aufgeteilt nach den vier Tastaturblöcken erstellt. Links finden Sie die Taste. Dabei bedeuten:

Esc = <Escape>
Bs = <Backspace>
Tab = <Tabulator>
Del = <Delete>
Ret = <Return>
Spc = <Space> (Leertaste)
Hlp = <Help>
Und = <Undo>
Ins = <Insert>
Cup = <Pfeil hoch> (Cursor UP)
Clr = <Clr/Home>
Clf = <Pfeil links> (Cursor LeFt)
Cdn = <Pfeil runter> (Cursor DowN)
Crt = <Pfeile rechts> (Cursor RighT)
Ent = <Enter>

Die erste Zahl in jeder Spalte ist der »Scancode, die zweite bezeichnet den »ASCII-Code. Alle Zahlen sind hexadezimal. Das "$"-Zeichen haben wir uns erspart, um die Tabelle übersichtlicher zu gestalten.

Welches Zeichen aus dem Zeichensatz zu einem bestimmten »ASCII-Code gehört, können Sie der Abb. 3 entnehmen. Die erste Hex-Ziffer des Codes bezeichnet die Reihe in der Abbildung, die zweite Ziffer die Spalte. Bei "$41" finden Sie in der 4. Reihe in der Spalte 1 (die linke Spalte ist Spalte 0) das "A".

Die Spaltenbezeichnungen haben folgende Bewandtnis:

o.S. = ohne Shift - Taste alleine wurde gedrückt
+S. = Taste und Shift wurden gleichzeitig gedrückt
+C. = Taste und Control gleichzeitig gedrückt
+S.+C. = Taste, Shift und Control gleichzeitig gedrückt
+A. = Taste und Alternate gleichzeitig gedrückt

Zunächst die obere Tastenreihe des Hauptblocks:

	o. S.	+S.	+C.	+S.+C.	+A.
Esc	1 - 1B	1 - 1B	1 - 1B	1 - 1B	1 - 1B
1	2 - 31	2 - 21	2 - 11	2 - 1	78 - 0
2	3 - 32	3 - 22	3 - 0	3 - 2	79 - 0
3	4 - 33	4 - DD	4 - 13	4 - 1D	7A - 0
4	5 - 34	5 - 24	5 - 14	5 - 4	7B - 0
5	6 - 35	6 - 25	6 - 15	6 - 5	7C - 0
6	7 - 36	7 - 26	7 - 1E	7 - 6	7D - 0
7	8 - 37	8 - 2F	8 - 17	8 - F	7E - 0
7	9 - 38	9 - 28	9 - 18	9 - 8	7F - 0
9	A - 39	A - 29	A - 19	A - 9	80 - 0
0	B - 30	B - 3D	B - 10	B - 1D	81 - 0
ß	C - 9E	C - 3F	C - 1E	C - 1F	82 - 0
'	D - 27	D - 60	D - 7	D - 0	83 - 0
#	29 - 23	29 - 5E	29 - 3	29 - 1E	29 - 23
Bs	E - 8	E - 8	E - 8	E - 8	E - 8

Zweite Tastaturreihe:

	o. S.	+S.	+C.	+S.+C.	+A.
Tab	F - 9	F - 9	F - 9	F - 9	F - 9
q	10 - 71	10 - 51	10 - 11	10 - 11	10 - 0
w	11 - 77	11 - 57	11 - 17	11 - 17	11 - 0
e	12 - 65	12 - 45	12 - 5	12 - 5	12 - 0
r	13 - 72	13 - 52	13 - 12	13 - 12	13 - 0
t	14 - 74	14 - 54	14 - 14	14 - 14	14 - 0
z	15 - 7A	15 - 5A	15 - 1A	15 - 1A	15 - 0
u	16 - 75	16 - 55	16 - 15	16 - 15	16 - 0
i	17 - 69	17 - 49	17 - 9	17 - 9	17 - 0
o	18 - 6F	18 - 4F	18 - F	18 - F	18 - 0
p	19 - 70	19 - 50	19 - 10	19 - 10	19 - 0
ü	1A - 81	1A - 9A	1A - 1	1A - 1A	1A - 40
+	1B - 2B	1B - 2A	1B - B	1B - A	1B - 2B
Del	53 - 7F	53 - 7F	53 - 1F	53 - 1F	53 - 7F

Dritte Tastaturreihe von oben:

	o. S.	+S.	+C.	+S.+C.	+A.
a	1E - 61	1E - 41	1E - 1	1E - 1	1E - 0
s	1F - 73	1F - 53	1F - 13	1F - 13	1F - 0
d	20 - 64	20 - 44	20 - 4	20 - 4	20 - 0
f	21 - 66	21 - 46	21 - 6	21 - 6	21 - 0
g	22 - 67	22 - 47	22 - 7	22 - 7	22 - 0
h	23 - 68	23 - 48	23 - 8	23 - 8	23 - 0
j	24 - 6A	24 - 4A	24 - A	24 - A	24 - 0
k	25 - 6B	25 - 4B	25 - B	25 - B	25 - 0
l	26 - 6C	26 - 4C	26 - C	26 - C	26 - 0
ö	27 - 94	27 - 99	27 - 14	27 - 19	27 - 5B
ä	28 - 84	28 - 8E	28 - 4	28 - E	28 - 5D
Ret	1C - D	1C - D	1C - A	1C - A	1C - D
~	2B - 7E	2B - 7C	2B - 1E	2B - 1C	2B - 7E

Vierte Tastaturreihe:

	o. S.	+S.	+C.	+S.+C.	+A.
<	60 - 3C	60 - 3E	60 - 1C	60 - 1E	60 - 3C
y	2C - 79	2C - 59	2C - 19	2C - 19	2C - 0
x	2D - 78	2D - 58	2D - 18	2D - 18	2D - 0
c	2E - 63	2E - 43	2E - 3	2E - 3	2E - 0
v	2F - 76	2F - 56	2F - 16	2F - 16	2F - 0
b	30 - 62	30 - 42	30 - 2	30 - 2	30 - 0
n	31 - 6E	31 - 4E	31 - E	31 - E	31 - 0
m	32 - 6D	32 - 4D	32 - D	32 - D	32 - 0
,	33 - 2C	33 - 3B	33 - C	33 - 1B	33 - 2C
.	34 - 2E	34 - 3A	34 - E	34 - 1A	34 - 2E
-	35 - 2D	35 - 5F	35 - 1F	35 - 1F	35 - 2D
Spc	39 - 20	39 - 20	39 - 0	39 - 0	39 - 20

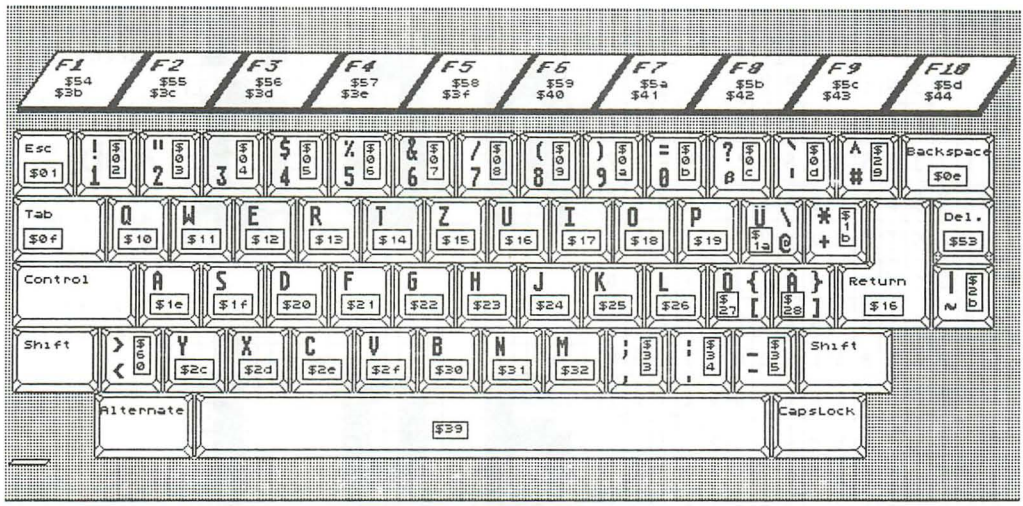
Cursor-Block:

	o. S.	+S.	+C.	+S.+C.	+A.
Hlp	62 - 0	62 - 0	62 - 0	62 - 0	(Hardcopy)
Und	61 - 0	61 - 0	61 - 0	61 - 0	
Ins	52 - 0	52 - 30	52 - 0	52 - 10	(Maustaste)
Cup	48 - 0	48 - 38	48 - 0	48 - 18	
Clr	47 - 0	47 - 37	77 - 0	77 - 37	
Clf	4B - 0	4B - 34	73 - 0	73 - 0	
Cdn	50 - 0	50 - 32	50 - 0	50 - 0	
Crt	4D - 0	4D - 36	74 - 0	74 - 0	

Ziffernblock:

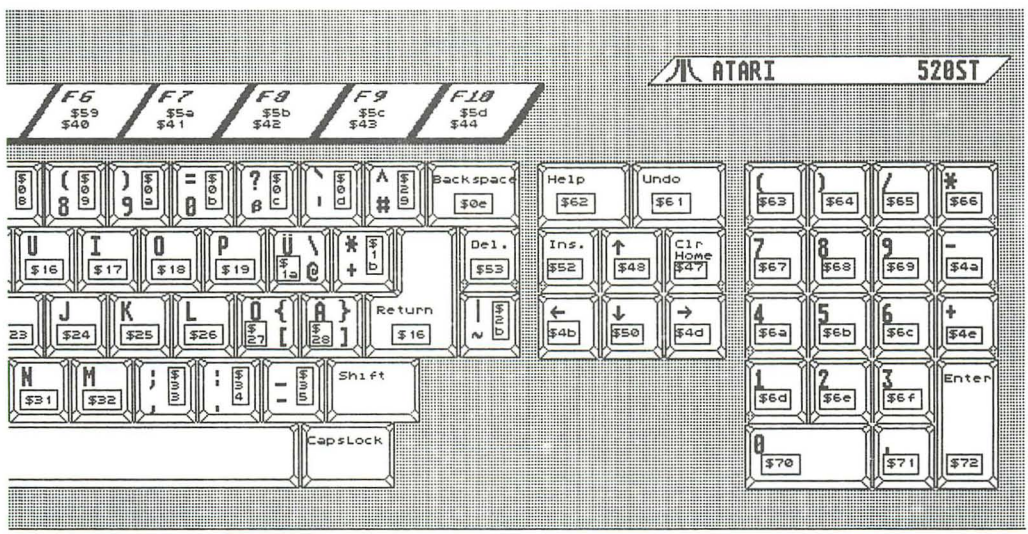
	o. S.	+S.	+C.	+S.+C.	+A.
(	63 - 28	63 - 28	63 - 8	63 - 8	63 - 28
)	64 - 29	64 - 29	64 - 9	64 - 9	64 - 29
/	65 - 2F	65 - 2F	65 - F	65 - F	65 - 2F
*	66 - 2A	66 - 2A	66 - A	66 - A	66 - 2A
7	67 - 37	67 - 37	67 - 17	67 - 17	67 - 37
8	68 - 38	68 - 38	68 - 18	68 - 18	68 - 38
9	69 - 39	69 - 39	69 - 19	69 - 19	69 - 39
-	4A - 2D	4A - 2D	4A - 1F	4A - 1F	4A - 2D
4	6A - 34	6A - 34	6A - 14	6A - 14	6A - 34
5	6B - 35	6B - 35	6B - 15	6B - 15	6B - 35
6	6C - 36	6C - 36	6C - 1E	6C - 1E	6C - 36
+	4E - 2B	4E - 2B	4E - B	4E - B	4E - 2B
1	6D - 31	6D - 31	6D - 11	6D - 11	6D - 31
2	6E - 32	6E - 32	6E - 0	6E - 0	6E - 32
3	6F - 33	6F - 33	6F - 13	6F - 13	6F - 33
Ent	72 - D	72 - D	72 - A	72 - A	72 - D
0	70 - 30	70 - 30	70 - 10	70 - 10	70 - 30
.	71 - 2E	71 - 2E	71 - E	71 - E	71 - 2E

Die Funktionstasten geben nur den »Scancode zurück. Sehen Sie dazu bitte Abb.. 1



Scancode der Tastatur

Abb. 1



Scancode der Tastatur, 2. Teil

Abb. 2

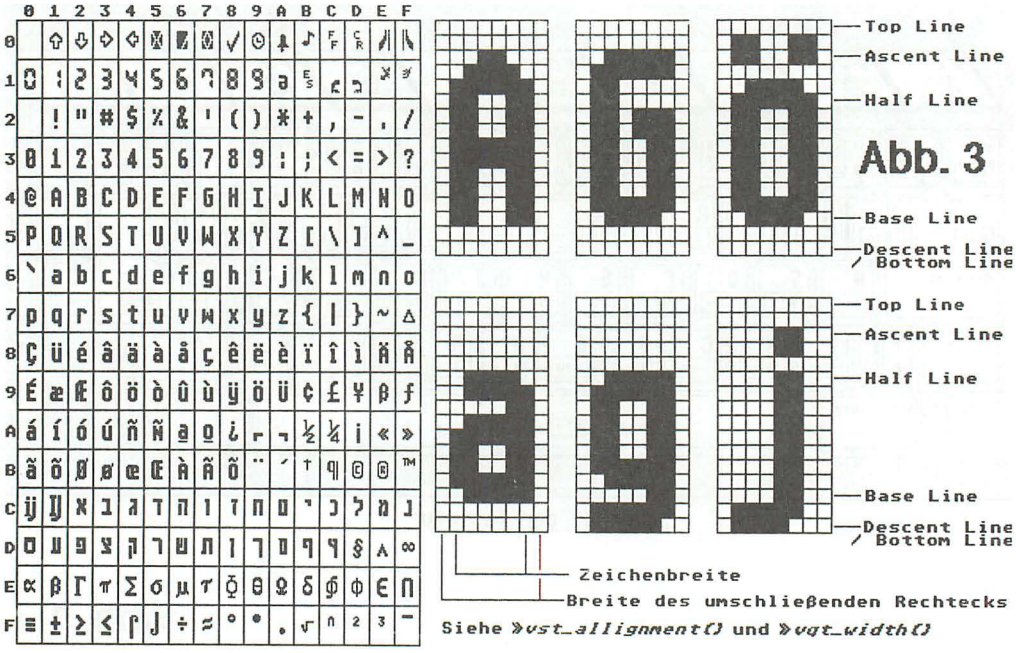


Abb. 3

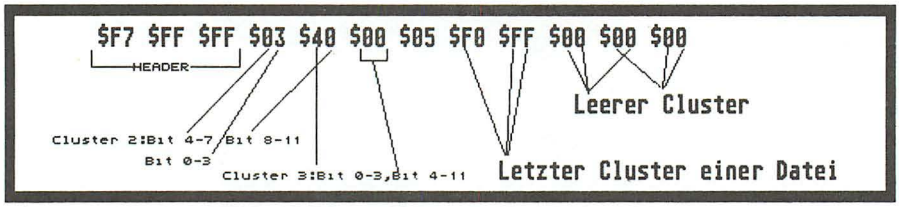
Siehe »*vst_allignment()* und »*vqt_width()*

Anhang E

In diesem Anhang finden Sie den grafischen Aufbau der »FAT auf der Diskette und eines »Tracks auf derselben.

FAT - Aufbau:

Ein kleiner Ausschnitt:



Ein Eintrag eines Cluster zeigt immer auf den nächsten Cluster der Datei!

Abb. 1

Trackaufbau bei einfacher Dichte

für 128 Byte Sektoren

Nur am Anfang vom Track

Dieses Stück wird für jeden Sektor wiederholt

40 Bytes '4E'	6 Bytes '00'	ID 'FE'	Track #	Seite #	Sektor #	Länge #	CRC 1	CRC 2	11 Bytes 'FF'	6 Bytes '00'	ID 'FB'	128 Byte Sektor	CRC 1	CRC 2	10 Bytes 'FF'

Abb. 2

Trackaufbau bei doppelter Dichte

für 256, 512 und 1024 Byte Sektoren

Nur am Anfang vom Track

Dieses Stück wird für jeden Sektor wiederholt

60 Bytes '4E'	12 Bytes '00'	3 Bytes 'A1'	ID 'FE'	Track #	Seite #	Länge #	CRC 1	CRC 2	22 Bytes '4E'	12 Bytes '00'	3 Bytes 'A1'	ID 'FE'	Sektor 256 Bytes 512 Bytes 1024 Bytes	CRC 1	CRC 2	24 Bytes '4E'

Abb. 3

Anhang F

Dieser Anhang enthält nummerisch geordnete Tabellen, der Funktionen vom »Bios, »Extended Bios, »Gemdos und »VDI.

Bios:

Funktionsnr.	Funktionsaufruf
$00 = 0	Getmpb(p_mpb)
$01 = 1	status = Bconstat(dev)
$03 = 3	Bconout(dev, c)
$04 = 4	error = Rwabs(rwflag, buf, count, recno, dev)
$05 = 5	oldvec = Setexc(vecnum, vec)
$06 = 6	millisec = Tickcal()
$07 = 7	zeiger = Getbpb(dev)
$08 = 8	status = Bcostat(dev)
$09 = 9	status = Mediach(dev)
$0A = 10	drvbits = Drvmap()
$0B = 11	oldmode = Kbshift(mode)

Extended Bios:

Funktionsnr.	Funktionsaufruf
$00 = 0	Initmouse(type, param, vec)
$01 = 1	zeiger = Ssbrk(amount)
$02 = 2	zeiger = Physbase()
$03 = 3	zeiger = Logbase()
$04 = 4	rez = Getrez()
$05 = 5	Setscreen(soft, phys, rez)
$06 = 6	Setpallete(palette)
$07 = 7	oldcol = Setcolor(register, color)
$08 = 8	error = Floprd(buf, filler, devno, sectno, trackno, sideno, count)
$09 = 9	error = Flopwr(buf, filler, devno, sectno, trackno, sideno, count)
$0A = 10	error = Flopfmt(buf, filler, devno, spt, trackno, sideno, interlv, magic, virgin)
$0B = 11	used-by-BIOS()
$0C = 12	Midiws(cnt, ptr)
$0D = 13	Mfpint(interno, vector)
$0E = 14	zeiger = Iorec(devno)
$0F = 15	Rsonf(speed, flowctl, ucr, rsr, tsr, scr)
$10 = 16	keytab = Keytbl(unshift, shift, capslock)
$11 = 17	zufall = Random()
$12 = 18	Protobt(buf, serialno, disktype, execflag)
$13 = 19	error = Flopver(buf, filler, devno, sectno, trackno, sideno, count)

Funktionsnr.	Funktionsaufruf
$14 = 20	Scrdmp()
$15 = 21	rate = Cursconf(function, operand)
$16 = 22	Settime(datetime)
$17 = 23	datetime = Gettime()
$18 = 24	Bioskeys()
$19 = 25	Ikbdws(cnt, ptr)
$1A = 26	Jdisint(intno)
$1B = 27	Jenabint(intno)
$1C = 28	ret = Giaccess(data, regno)
$1D = 29	Offgibit(bitno)
$1E = 30	Ongibit(bitno)
$1F = 31	Xbtimer(timer, control, data, vec)
$20 = 32	Dosound(ptr)
$21 = 33	oldconf = Setprt(config)
$22 = 34	zeiger = Kbdvbase()
$23 = 35	rate = Kbrate(initial, repeat)
$24 = 36	Prtblk(zeiger)
$25 = 37	Vsync()
$26 = 38	Supexec(codeptr)
$27 = 39	Puntaes()

Gemdos:

Funktionsnr.	Funktionsaufruf
$00 = 0	Pterm0()
$01 = 1	zeichen = Cconin()
$02 = 2	Cconout(chr)
$03 = 3	zeichen = Cauxin()
$04 = 4	Cauxout(chr)
$05 = 5	Cprnout(chr)
$06 = 6	Crawio(wrd)
$07 = 7	zeichen = Crawcin()
$08 = 8	zeichen = Cnecin()
$09 = 9	Cconws(ptr)
$0A = 10	Cconrs(buf)
$0B = 11	status = Cconis()
$0E = 14	olddrv = Dsetdrv(drv)
$10 = 16	status = Cconos()
$11 = 17	status = Cprnos()
$12 = 18	status = Cauxis()
$13 = 19	status = Cauxout(chr)
$19 = 25	drv = Dgetdrv()
$1A = 26	Fsetdta(dta)
$20 = 32	oldstack = Super(stack)
$2A = 42	date = Tgetdate()
$2B = 43	Tsetdate(date)
$2C = 44	time = Tgetime()

Funktionsnr.	Funktionsaufruf
$2D = 45	Tsettime(time)
$2F = 47	dta = Fgetdta()
$30 = 48	version = Sversion()
$31 = 49	Ptermres(n_bytes, rc)
$36 = 54	error = Dfree(pbuffer, drv)
$39 = 57	error = Dcreate(path)
$3A = 58	error = Ddelete(path)
$3B = 59	error = Dsetpath(path)
$3C = 60	handle = Fcreate(name, attr)
$3D = 61	handle = Fopen(pname, mode)
$3E = 62	error = Fclose(handle)
$3F = 63	error = Fread(handle, count, buf)
$40 = 64	error = Fwrite(handle, count, buf)
$41 = 65	error = Fdelete(name)
$42 = 66	error = Fseek(softs, handle, smode)
$43 = 67	attr = Fattrib(pname, wrt, mode)
$45 = 69	handle = Fdup(stdhnd)
$46 = 70	error = Fforce(stdhnd, nsthnd)
$47 = 71	error = Dgetpath(pathbuf, drv)
$48 = 72	adr = Malloc(nbytes)
$49 = 73	error = Mfree(adr)
$4A = 74	error = Mshrink(res, mp, size)
$4B = 75	base = Pexec(load, pcspec, pcmdln, penvstr)
$4C = 76	Pterm(code)
$4E = 78	error = Fsfirst(name, attr)
$4F = 79	error = Fsnext()
$56 = 86	error = Frename(res, oldname, newname)
$57 = 87	Fdatime(handle, buf, set)

VDI:

Funktionsnr.	Funktionsaufruf
$01 = 1	v_opnwk(work_in, &handle, work_out)
$02 = 2	v_clswk(handle)
$03 = 3	v_clrwk(handle)
$04 = 4	v_updwk(handle)
$05 = 5	v_bit_image(handle, filename, aspect, x_scale, y_scale, h_align, v_align, xyarray)
$05 = 5	v_clear_disp_list(handle)
$05 = 5	v_rmcur(handle)
$05 = 5	v_write_meta(handle, num_intin, intin, num_ptsin, ptsin)
$05 = 5	v_output_window(handle, xyarray)
$05 = 5	v_curaddress(handle, row, columns)
$05 = 5	v_curdown(handle)
$05 = 5	v_curhome(handle)
$05 = 5	v_curleft(handle)

Funktionsnr.	Funktionsaufruf
$05 = 5	v_curright(handle)
$05 = 5	v_curtext(handle, &string)
$05 = 5	v_curup(handle)
$05 = 5	v_dspcur(handle, x, y)
$05 = 5	v_eeol(handle)
$05 = 5	v_eeos(handle)
$05 = 5	status = vq_tabstatus(handle)
$05 = 5	v_rvon(handle)
$05 = 5	v_rvoff(handle)
$05 = 5	v_enter_cur(handle)
$05 = 5	v_exit_cur(handle)
$05 = 5	vm_filename(handle, filename)
$05 = 5	v_form_adv(handle)
$05 = 5	v_hardcopy(handle)
$05 = 5	v_meta_extents(handle, min_x, min_y, max_x, max_y)
$05 = 5	v_chcells(handle, &rows, &columns)
$06 = 6	v_pline(handle, count, pxyarray)
$07 = 7	v_pmarker(handle, count, pxyarray)
$08 = 8	v_gtext(handle, x, y, string)
$09 = 9	v_fillarea(handle, count, pxyarray)
$0A = 10	v_cellarray(handle, pxyarray, row_length, el_used, num_rows, wrt_mode, colarray)
$0B = 11	v_bar(handle, pxyarray)
$0B = 11	v_pieslice(handle, x, y, radius, begang, endang)
$0B = 11	v_justified(handle, x, y, string, length, word_space, char_space)
$0B = 11	v_ellarc(handle, x, y, xradius, yradius, begang, endang)
$0B = 11	v_rfbox(handle, xyarray)
$0B = 11	v_rbox(handle, xyarray)
$0B = 11	v_circle(handle, x, y, radius)
$0B = 11	v_ellipse(handle, x, y, xradius, yradius)
$0B = 11	v_ellpie(handle, x, y, xradius, yradius, begang, endang)
$0B = 11	v_arc(handle, x, y, radius, begang, endang)
$0C = 12	vst_height(handle, height, &char_width, &char_height &cell_width, &cell_height)
$0D = 13	set_baseline = vst_rotation(handle, angle)
$0E = 14	vs_color(handle, index, rgb_in)
$0F = 15	set_type = vsl_type(handle, style)
$10 = 16	set_width = vsl_width(handle, width)
$11 = 17	set_color = vsl_color(handle, color_index)
$12 = 18	set_type = vsm_type(handle, symbol)
$13 = 19	set_height = vsm_height(handle, height)
$14 = 20	set_color = vsm_color(handle, color_index)
$15 = 21	set_font = vst_font(handle, font)
$16 = 22	set_color = vst_color(handle, color_index)
$17 = 23	set_interior = vsf_interior(handle, style)

Funktionsnr.	Funktionsaufruf
$18 = 24	set_style = vsf_style(handle, style_index)
$19 = 25	set_color = vsf_color(handle, color_index)
$1A = 26	vq_color(handle, color_index, set_flag, rgb)
$1B = 27	vq_cellarray(handle, pxyarray, row_length, num_rows, &el_used, &rows_used, &status, colarray)
$1C = 28	status = vsm_locator(handle, x, y, &xout, &yout, &term)
$1C = 28	vrq_locator(handle, x, y, &xout, &yout, &term)
$1D = 29	vsm_valuator(handle, val_in, &val_out, &term, &status)
$1D = 29	vrq_valuator(handle, valuator_in, &valuator_out, &terminator)
$1E = 30	status = vsm_choice(handle, &choice)
$1E = 30	status = vsm_choice(handle, ch_in, &ch_out)
$1F = 31	vrq_string(handle, max_length, echo_mode, echo_xy, &string)
$1F = 31	vsm_string(handle, max_length, echo_mode, echo_xy, &string)
$20 = 32	set_mode = vswr_mode(handle, mode)
$21 = 33	vsin_mode(handle, dev_type, mode)
$23 = 35	vql_attributes(handle, attrib)
$24 = 36	vqm_attributes(handle, attrib)
$25 = 37	vqf_attributes(handle, attrib)
$26 = 38	vqt_attributes(handle, attrib)
$27 = 39	vst_alignment(handle, hor_in, vert_in, &hor_out, &vert_out)
$64 = 100	v_opnvwk(work_in, &handle, work_out)
$65 = 101	v_clsvwk(handle)
$66 = 102	vq_extnd(handle, owflag, work_out)
$67 = 103	v_contourfill(handle, x, y, index)
$68 = 104	set_perimeter = vsf_perimeter(handle, per_vis)
$69 = 105	v_get_pixel(handle, x, y, &pel, &index)
$6A = 106	set_effects = vst_effects(handle, effect)
$6B = 107	set_point = vst_point(handle, point, &char_width, &char_height &cell_width, &cell_height)
$6C = 108	vsl_ends(handle, beg_style, end_style)
$6D = 109	vro_cpyfm(handle, wr_mode, pxyarray, psrcMFDB, pdesMFDB)
$6E = 110	vr_trnfm(handle, psrcMFDB, pdesMFDB)
$6F = 111	vsc_form(handle, pcur_form)
$70 = 112	vsf_udpat(handle, pfill_pat, planes)
$71 = 113	vsl_udsty(handle, pattern)
$72 = 114	vr_recfl(handle, pxyarray)
$73 = 115	vqin_mode(handle, dev_type, &input_mode)
$74 = 116	vqt_extent(handle, string, extent)

Funktionsnr.	Funktionsaufruf
$75 = 117	status = vtq_width(handle, character, &cell_width, &left_delta, &right_delta)
$76 = 118	vex_timv(handle, tim_addr, otim_addr, &tim_conv)
$77 = 119	additional = vst_load_fonts(handle, select)
$78 = 120	vst_unload_fonts(handle, select)
$79 = 121	vrt_cpyfm(handle, wr_mode, pxyarray, psrcMFDB, pdesMFDB, color_index)
$7A = 122	v_show_c(handle, reset)
$7B = 123	v_hide_c(handle)
$7C = 124	vq_mouse(handle, &pstatus, &x, &y)
$7D = 125	vex_butv(handle, pusrcode, psavcode)
$7E = 126	vex_motv(handle, pusrcode, psavcode)
$7F = 127	vex_curv(handle, pusrcode, psavcode)
$80 = 128	vq_key_s(handle, &pstatus)
$81 = 129	vs_clip(handle, clip_flag, pxyarray)
$82 = 130	index = vqt_name(handle, element_num, name)
$83 = 131	vqt_fontinfo(handle, &minADE, &maxADE, distances, &maxwidth, effects)

Anhang G

Dieser Anhang enthält die Funktionen vom »Bios, »Extended Bios, »Gemdos und »VDI nach Alphabet geordnet.

Bios:

Funktionsnr.	Funktionsaufruf
$01 = 1	status = Bconstat(dev)
$08 = 8	status = Bcostat(dev)
$0A = 10	drvbits = Drvmap()
$07 = 7	zeiger = Getbpb(dev)
$00 = 0	Getmpb(p_mpb)
$0B = 11	oldmode = Kbshift(mode)
$09 = 9	status = Mediach(dev)
$04 = 4	error = Rwabs(rwflag, buf, count, recno, dev)
$05 = 5	oldvec = Setexc(vecnum, vec)
$06 = 6	millisec = Tickcal()

Extended Bios:

Funktionsnr.	Funktionsaufruf
$18 = 24	Bioskeys()
$15 = 21	rate = Cursconf(function, operand)
$20 = 32	Dosound(ptr)
$0A = 10	error = Flopfmt(buf, filler, devno, spt, trackno, sideno, interlv, magic, virgin)
$08 = 8	error = Floprd(buf, filler, devno, sectno, trackno, sideno, count)
$13 = 19	error = Flopver(buf, filler, devno, sectno, trackno, sideno, count)
$09 = 9	error = Flopwr(buf, filler, devno, sectno, trackno, sideno, count)
$04 = 4	rez = Getrez()
$17 = 23	datetime = Gettime()
$1C = 28	ret = Giaccess(data, regno)
$19 = 25	Ikbdws(cnt, ptr)
$00 = 0	Initmouse(type, param, vec)
$0E = 14	zeiger = Iorec(devno)
$1A = 26	Jdisint(intno)
$1B = 27	Jenabint(intno)
$22 = 34	zeiger = Kbdvbase()
$23 = 35	rate = Kbrate(initial, repeat)
$10 = 16	keytab = Keytbl(unshift, shift, capslock)
$03 = 3	zeiger = Logbase()
$0D = 13	Mfpint(interno, vector)
$0C = 12	Midiws(cnt, ptr)

Funktionsnr.	Funktionsaufruf
$1D = 29	Offgibit(bitno)
$1E = 30	Ongibit(bitno)
$02 = 2	zeiger = Physbase()
$12 = 18	Protobt(buf, serialno, disktype, execflag)
$24 = 36	Prtblk(zeiger)
$27 = 39	Puntaes()
$11 = 17	zufall = Random()
$0F = 15	Rsonf(speed, flowctl, ucr, rsr, tsr, scr)
$14 = 20	Scrdmp()
$07 = 7	oldcol = Setcolor(register, color)
$06 = 6	Setpallete(palette)
$21 = 33	oldconf = Setprt(config)
$05 = 5	Setscreen(soft, phys, rez)
$16 = 22	Settime(datetime)
$01 = 1	zeiger = Ssbrk(amount)
$26 = 38	Supexec(codeptr)
$0B = 11	used-by-BIOS()
$25 = 37	Vsync()

Gemdos:

Funktionsnr.	Funktionsaufruf
$03 = 3	zeichen = Cauxin()
$12 = 18	status = Cauxis()
$13 = 19	status = Cauxout(chr)
$04 = 4	Cauxout(chr)
$01 = 1	zeichen = Cconin()
$0B = 11	status = Cconis()
$10 = 16	status = Cconos()
$02 = 2	Cconout(chr)
$0A = 10	Cconrs(buf)
$09 = 9	Cconws(ptr)
$08 = 8	zeichen = Cnecin()
$11 = 17	status = Cprnos()
$05 = 05	Cprnout(chr)
$07 = 7	zeichen = Crawcin()
$06 = 6	Crawio(wrd)
$39 = 57	error = Dcreate(path)
$3A = 58	error = Ddelete(path)
$36 = 54	error = Dfree(pbuffer, drv)
$19 = 25	drv = Dgetdrv()
$47 = 71	error = Dgetpath(pathbuf, drv)
$0E = 14	olddrv = Dsetdrv(drv)
$3B = 59	error = Dsetpath(path)
$43 = 67	attr = Fattrib(pname, wrt, mode)
$3E = 62	error = Fclose(handle)
$3C = 60	handle = Fcreate(name, attr)

Funktionsnr.	Funktionsaufruf
$57 = 87	Fdatime(handle, buf, set))
$41 = 65	error = Fdelete(name)
$45 = 69	handle = Fdup(stdhnd)
$46 = 70	error = Fforce(stdhnd, nsthnd)
$2F = 47	dta = Fgetdta()
$3D = 61	handle = Fopen(pname, mode)
$3F = 63	error = Fread(handle, count, buf)
$56 = 86	error = Frename(res, oldname, newname)
$42 = 66	error = Fseek(softs, handle, smode)
$1A = 26	Fsetdta(dta)
$4E = 78	error = Fsfirst(name, attr)
$4F = 79	error = Fsnext()
$40 = 64	error = Fwrite(handle, count, buf)
$48 = 72	adr = Malloc(nbytes)
$49 = 73	error = Mfree(adr)
$4A = 74	error = Mshrink(res, mp, size)
$4B = 75	base = Pexec(load, pcspec, pcmdln, penvstr)
$00 = 0	Pterm0()
$4C = 76	Pterm(code)
$31 = 49	Ptermres(n_bytes, rc)
$20 = 32	oldstack = Super(stack)
$30 = 48	version = Sversion()
$2A = 42	date = Tgetdate()
$2C = 44	time = Tgetime()
$2B = 43	Tsetdate(date)
$2D = 45	Tsettime(time)

VDI:

Funktionsnr.	Funktionsaufruf
$0B = 11	v_arc(handle, x, y, radius, begang, endang)
$0B = 11	v_bar(handle, pxyarray)
$05 = 5	v_bit_image(handle, filename, aspect, x_scale, y_scale, h_align, v_align, xyarray)
$0A = 10	v_cellarray(handle, pxyarray, row_length, el_used, num_rows, wrt_mode, colarray)
$05 = 5	v_chcells(handle, &rows, &columns)
$0B = 11	v_circle(handle, x, y, radius)
$05 = 5	v_clear_disp_list(handle)
$03 = 3	v_clrwk(handle)
$65 = 101	v_clsvwk(handle)
$02 = 2	v_clswk(handle)
$67 = 103	v_contourfill(handle, x, y, index)
$05 = 5	v_curaddress(handle, row, columns)
$05 = 5	v_curdown(handle)
$05 = 5	v_curhome(handle)
$05 = 5	v_curleft(handle)

Funktionsnr.	Funktionsaufruf
$05 = 5	v_curright(handle)
$05 = 5	v_curtext(handle, &string)
$05 = 5	v_curup(handle)
$05 = 5	v_dspcur(handle, x, y)
$05 = 5	v_eeol(handle)
$05 = 5	v_eeos(handle)
$0B = 11	v_ellarc(handle, x, y, xradius, yradius, begang, endang)
$0B = 11	v_ellipse(handle, x, y, xradius, yradius)
$0B = 11	v_ellpie(handle, x, y, xradius, yradius, begang, endang)
$05 = 5	v_enter_cur(handle)
$05 = 5	v_exit_cur(handle)
$09 = 9	v_fillarea(handle, count, pxyarray)
$05 = 5	v_form_adv(handle)
$69 = 105	v_get_pixel(handle, x, y, &pel, &index)
$08 = 8	v_gtext(handle, x, y, string)
$05 = 5	v_hardcopy(handle)
$7B = 123	v_hide_c(handle)
$0B = 11	v_justified(handle, x, y, string, length, word_space, char_space)
$05 = 5	v_meta_extents(handle, min_x, min_y, max_x, max_y)
$64 = 100	v_opnvwk(work_in, &handle, work_out)
$01 = 1	v_opnwk(work_in, &handle, work_out)
$05 = 5	v_output_window(handle, xyarray)
$0B = 11	v_pieslice(handle, x, y, radius, begang, endang)
$06 = 6	v_pline(handle, count, pxyarray)
$07 = 7	v_pmarker(handle, count, pxyarray)
$0B = 11	v_rbox(handle, xyarray)
$0B = 11	v_rfbox(handle, xyarray)
$05 = 5	v_rmcur(handle)
$05 = 5	v_rvoff(handle)
$05 = 5	v_rvon(handle)
$7A = 122	v_show_c(handle, reset)
$04 = 4	v_updwk(handle)
$05 = 5	v_write_meta(handle, num_intin, intin, num_ptsin, ptsin)
$7D = 125	vex_butv(handle, pusrcode, psavcode)
$7F = 127	vex_curv(handle, pusrcode, psavcode)
$7E = 126	vex_motv(handle, pusrcode, psavcode)
$76 = 118	vex_timv(handle, tim_addr, otim_addr, &tim_conv)
$05 = 5	vm_filename(handle, filename)
$1B = 27	vq_cellarray(handle, pxyarray, row_length, num_rows, &el_used, &rows_used, &status, colarray)
$1A = 26	vq_color(handle, color_index, set_flag, rgb)
$66 = 102	vq_extnd(handle, owflag, work_out)
$80 = 128	vq_key_s(handle, &pstatus)

Funktionsnr.	Funktionsaufruf
$7C = 124	vq_mouse(handle, &pstatus, &x, &y)
$05 = 5	status = vq_tabstatus(handle)
$25 = 37	vqf_attributes(handle, attrib)
$73 = 115	vqin_mode(handle, dev_type, &input_mode)
$23 = 35	vql_attributes(handle, attrib)
$24 = 36	vqm_attributes(handle, attrib)
$26 = 38	vqt_attributes(handle, attrib)
$74 = 116	vqt_extent(handle, string, extent)
$83 = 131	vqt_fontinfo(handle, &minADE, &maxADE, distances, &maxwidth, effects)
$82 = 130	index = vqt_name(handle, element_num, name)
$72 = 114	vr_recfl(handle, pxyarray)
$6E = 110	vr_trnfm(handle, psrcMFDB, pdesMFDB)
$6D = 109	vro_cpyfm(handle, wr_mode, pxyarray, psrcMFDB, pdesMFDB)
$1C = 28	vrq_locator(handle, x, y, &xout, &yout, &term)
$1F = 31	vrq_string(handle, max_length, echo_mode, echo_xy, &string)
$1D = 29	vrq_valuator(handle, valuator_in, &valuator_out, &terminator)
$79 = 121	vrt_cpyfm(handle, wr_mode, pxyarray, psrcMFDB, pdesMFDB, color_index)
$81 = 129	vs_clip(handle, clip_flag, pxyarray)
$0E = 14	vs_color(handle, index, rgb_in)
$6F = 111	vsc_form(handle, pcur_form)
$19 = 25	set_color = vsf_color(handle, color_index)
$17 = 23	set_interior = vsf_interior(handle, style)
$68 = 104	set_perimeter = vsf_perimeter(handle, per_vis)
$18 = 24	set_style = vsf_style(handle, style_index)
$70 = 112	vsf_udpat(handle, pfill_pat, planes)
$21 = 33	vsin_mode(handle, dev_type, mode)
$11 = 17	set_color = vsl_color(handle, color_index)
$6C = 108	vsl_ends(handle, beg_style, end_style)
$0F = 15	set_type = vsl_type(handle, style)
$71 = 113	vsl_udsty(handle, pattern)
$10 = 16	set_width = vsl_width(handle, width)
$1E = 30	status = vsm_choice(handle, &choice)
$1E = 30	status = vsm_choice(handle, ch_in, &ch_out)
$14 = 20	set_color = vsm_color(handle, color_index)
$13 = 19	set_height = vsm_height(handle, height)
$1C = 28	status = vsm_locator(handle, x, y, &xout, &yout, &term)
$1F = 31	vsm_string(handle, max_length, echo_mode, echo_xy, &string)
$12 = 18	set_type = vsm_type(handle, symbol)
$1D = 29	vsm_valuator(handle, val_in, &val_out, &term, &status)

Funktionsnr.	Funktionsaufruf
$27 = 39	vst_alignment(handle, hor_in, vert_in, &hor_out, &vert_out)
$16 = 22	set_color = vst_color(handle, color_index)
$6A = 106	set_effects = vst_effects(handle, effect)
$15 = 21	set_font = vst_font(handle, font)
$0C = 12	vst_height(handle, height, &char_width, &char_height &cell_width, &cell_height)
$77 = 119	additional = vst_load_fonts(handle, select)
$6B = 107	set_point = vst_point(handle, point, &char_width, &char_height &cell_width, &cell_height)
$0D = 13	set_baseline = vst_rotation(handle, angle)
$78 = 120	vst_unload_fonts(handle, select)
$20 = 32	set_mode = vswr_mode(handle, mode)
$75 = 117	status = vtq_width(handle, character, &cell_width, &left_delta, &right_delta)

Jürgen und Dieter Geiss

Hüthig

Software-Entwicklung auf dem Atari ST

Programmieren unter GEM und TOS

1986, 390 S., kart.,
DM 54,—
ISBN 3-7785-1339-7

Dieses Buch enthält alles, was ein ernsthafter Programmierer braucht, um gute und professionelle Software auf dem Atari ST zu entwickeln.

Dabei wird auf drei Programmiersprachen eingegangen: C, Pascal und Modula-2. Es ist für Anfänger und Fortgeschrittene gleichermaßen geeignet. Zunächst wird die Entwicklung von TOS-Programmen; das sind Programme, die auf Text-Ebene laufen, aufgezeigt. In diesem Rahmen wird das GEMDOS, BIOS und XBIOS mit Beispielen genau erklärt. Anschließend werden die beiden Teile des GEM, AES und VDI gründlich beschrieben. Darauf aufbauend wird die Entwicklung von GEM-Programmen beschrieben. Dabei wird zuerst der Umgang mit dem Resource-Construction-Set geübt und in zwei Lektionen die Arbeit mit diesem vertieft. Der daraus resultierende Objektbaum wird in seiner Struktur genau untersucht.

An zwei kompletten Beispielprogrammen, einer Applikation und einem Desk-Accessory werden die typischen Probleme der GEM-Programmierung gelöst. Dazu gehören auch die Verwaltung von Fenstern, Menüs und Dialogboxen. Der komfortable Drucker-Spooler ist ein fertiges Utility-Programm zum Drucken von Dateien.

Für das schnelle Auffinden von Systemprozeduren und Systemvariablen sind im Anhang ausführliche Tabellen aufgeführt.

Dr. Alfred Hüthig Verlag
Im Weiher 10
6900 Heidelberg 1

Hüthig

Dieter und Jürgen Geiss

Logo auf dem Atari ST

1986, 145 S., kart.,
DM 35,—
ISBN 3-7785-1262-5

LOGO ist die erste Sprache auf dem Atari ST. Hier treffen sich die hervorragenden grafischen Fähigkeiten einer Programmiersprache und die überlegenen Leistungen des neuen Rechners.

Das vorliegende Buch zeigt das Planen und Schreiben von faszinierenden und nützlichen Programmen. Es lehrt die gesamte LOGO-Sprache mit strukturierter Top-Down-Programmierung, Prozeduren, Rekursionen usw. In „Logo auf dem Atari ST" stehen die Antworten auf die Fragen, die im Original-Handbuch offen geblieben sind.

Einige beispielhafte Projekte zeigen, daß LOGO weit mehr ist als eine anschauliche Lernsprache für Kinder.

Dr. Alfred Hüthig Verlag
Im Weiher 10
6900 Heidelberg 1

Wie jedes Lexikon ist auch dieses vollständig nach Stichworten sortiert. Im Gegensatz zu einem normalen Lexikon findet der Leser hier jedoch nicht nur eine Beschreibung, sondern gleich eine Programmieranleitung.
Ebenso sind mehrere Tabellen enthalten, die das Auffinden weiterer Stichworte erleichtern und zusätzliche Informationen beinhalten. Die meisten anderen Bücher enthalten entweder eine Dokumentation über die GEM-Fähigkeiten des Rechners oder die Beschreibung der einfachen Betriebssystemroutinen. Hier jedoch findet der Leser alles. Es gibt nicht nur Hinweise zur Programmierung von Dialogboxen, Fenstern oder Kommandointerpretern, sondern es werden auch alle systeminternen Fragen beantwortet. Dies umfaßt sowohl die Programmierung der im Rechner benutzten Chips, als auch eine Beschreibung der Schnittstellen und deren Benutzung. Es wird auf alle grafischen Möglichkeiten des ST eingegangen. Gleichgültig, ob nach den deutschen oder nach den englischen Begriffen gesucht wird, es sind alle vorhanden und verweisen gegebenenfalls aufeinander.